Formal Analysis of Mobile Multi-Factor Authentication with Single Sign-On Login*

Giada Sciarretta[1], Roberto Carbone[1], Silvio Ranise[1], and Luca Viganò[2]

[1]Security & Trust, FBK, Trento, Italia, {giada.sciarretta,carbone,ranise}@fbk.eu
[2]Department of Informatics, King's College London, UK, luca.vigano@kcl.ac.uk

Abstract

Over the last few years, there has been an almost exponential increase in the number of mobile applications that deal with sensitive data, such as applications for e-commerce or health. When dealing with sensitive data, classical authentication solutions based on username-password pairs are not enough, and multi-factor authentication solutions that combine two or more authentication factors of different categories are required instead. Even if several solutions are currently used, their security analyses have been performed informally or semi-formally at best, and without a reference model and a precise definition of the multi-factor authentication property. This makes a comparison among the different solutions both complex and potentially misleading. In this paper, we first present the design of two reference models for native applications based on the requirements of two real-world use-case scenarios. Common features between them are the use of one-time password approaches and the support of a single sign-on experience. Then, we provide a formal specification of our threat model and the security goals, and discuss the automated security analysis that we performed. Our formal analysis validates the security goals of the two reference models we propose and provides an important building block for the formal analysis of different multi-factor authentication solutions.

1 Introduction

Context and Motivations. Over the last few years, there has been an almost exponential increase of the number of *native mobile applications* (or *apps*, for short) that deal with sensitive data, ranging from apps for e-commerce, banking and finance to apps for well-being and health. One of the main reasons behind such a success is that mobile apps considerably increase the portability and efficiency of online services. Banking apps allow users not only to check their account balances but also to move money and pay bills or friends [11]. Mobile health apps range from personal health records (PHR) to personal digital assistants using connected devices such as smartwatches and other body-worn devices or implants. There are nowadays more than 325,000 mobile health apps (of which 158,000 are Android apps) on the market, a number that is increasing on a weekly basis [45].

However, also the reports on security and privacy issues in mobile apps are increasing on a weekly basis, bearing concrete witness to the fact that the management of sensitive data is often not properly taken into account by the developers of the apps. For example, the studies performed by He et al. [25] on free mobile health apps available on the Google Play store show that the majority of these apps send sensitive data in clear text and store them on third party servers that do not support the required confidentiality measures.

When dealing with sensitive data, classical authentication solutions based on username-password pairs are not enough. The "General Data Protection Regulation" of the European Union [16] mandates that specific security measures must be implemented, including *Multi-Factor Authentication (MFA)*, a strong(er)

*To appear in ACM Transactions on Privacy and Security, 2020

authentication solution that combines two or more authentication factors of different categories (e.g., a password combined with a PIN sent to a mobile device, or some biometric data). There are many alternative solutions on the market for providing MFA, usually based on *One-Time Passwords (OTPs)*, which are passwords that are valid for a short time and can only be used once. Examples are FIDO (Fast IDentity Online, `https://fidoalliance.org`), which enables mobile devices to act as Universal 2nd Factor authentication devices over Bluetooth or NFC, and Mobile Connect (`https://mobileconnect.io`), which identifies users through their mobile phone numbers.

In addition to the establishment of high-level security for authentication solutions for mobile apps, it is essential to take the usability aspect into consideration. Monitoring apps often require a daily or even hourly use, but understandably users cannot be bothered by a long and complex authentication procedure each time they want to read or update their data, especially on mobile devices where the keyboard is small and sometimes uncomfortable to use. A better usability can be provided by supporting a *Single Sign-On (SSO) experience*, which allows users to access different, federated apps by performing a single login, carried out with a selected identity provider (e.g., Facebook or Google). While the authenticated session is valid, users can directly access all the apps in the federation, without having to enter their credentials again and again.

Contributions. In this paper, we focus on MFA solutions for Android[1] apps based on OTPs and the support of a federated SSO experience.

Many alternative MFA solutions are available on the market. For instance, Google Authenticator is a mobile app that generates OTPs [23]. Like Google Authenticator, many of the OTP-generation solutions on the market are applicable only to web solutions and use mobile devices as an additional factor. In the scenario considered in this paper, users are not accessing web apps on their laptop or desktop, instead they are accessing native mobile apps. Even if there are some solutions currently used, to the best of our knowledge, their security analyses have been performed informally or semi-formally at best, and without a precise definition of the MFA property that they are supposed to satisfy. This makes a comparison between the different solutions both complex and potentially misleading. The main difficulty is probably the unclear and subtle detection of which compromised authentication factors lead to breaking the protocol security.

As a first significant step to overcome these problems, we make four main contributions:

1. We have designed two MFA reference models[2] for native apps based on the requirements of two real-world use-case scenarios, namely the use of a native app as OTP generator and the use of SAML 2.0 as authentication protocols. The common features between these two MFA reference models are the use of OTP approaches and the support of a SSO experience.

2. We have formulated a description of the proposed reference models detailing the security and trust assumptions given by different implementation choices.

3. We have introduced the concept of instance factors, that lets us formally define the threat model (specifying how and which instance factors are explicitly compromised) and the security property that a MFA solution must guarantee.

4. We have performed a two-level security analysis. First, we have identified the instance factors that are *explicitly* compromised (e.g., by stealing an ownership factor) by the different capabilities of an attacker. More complex attack traces, where the attacker *implicitly* compromises other instance factors exploiting some vulnerabilities of the protocol (e.g., by convincing users to finalize, using their own factors, an authentication started on the attacker's device) cannot be easily detected. Hence, in the second level, we have formally analyzed our reference models by modeling the flow, assumptions, attackers and goals using a formal language (ASLan++) and model-checking the security goal with the SATMC tool.

[1] We focus on Android for practical reasons, but our approach can be similarly applied to other operating systems such as iOS.

[2] A reference model is an abstract, semi-formal, representation of an authentication flow.

Even if authentication and authorization are strictly related, the authorization aspect is out of the scope of this paper. We decided to analyze MFA in the context of federated SSO login as a requirement given by our use-case scenarios. However, it is important to note that by doing so, we do not lose generality given that federated authentication systems are widespread scenarios. Moreover, since federated authentication is more complex than authentication on individual domains, our analysis could quite straightforwardly be customized to support MFA in that simplified context.

Our work is based on [47], where we have presented a MFA solution relying on the use of the *Time-based OTP (TOTP)* generation approach [28] and formally analyzed the proposed reference model in the context of a real-world scenario involving mobile e-health apps. In this paper, we extend [47] along the following lines:

1. We have generalized the description of the solution analyzed in [47] and provided the design of an additional reference model based on the use of the *Challenge-Response (CR)* approach [27] and a hardware token. This reference model is based on the requirements given by a real-world scenario involving the Italian national electronic identity card, which complies with the European standard of electronic identity documents.

2. We have refined the formalization of a MFA solution: instead of identifying a set of *strong* and *weak* assumptions as done in [47] to characterize the role of multi factors in the authentication process, we evaluate the effectiveness of MFA solutions against a set of attackers with different capabilities. This makes the result of the analysis more understandable and concrete.

3. We have provided the description and formalization of a set of proximity and hacker attackers in terms of the operations that they can perform (e.g., get to know a PIN value or steal the user smartphone) and which factors are compromised.

4. We have formally analyzed the two reference models based on the new formalization using the SATMC tool.

It is worth to mention that, for concreteness, we have developed our ideas in the context of two real use-case scenarios (mobile health apps and public administration online services). However, our reference models are composed of a number of building-blocks that can be combined in different ways to support a broader set of scenarios, and our threat models and formalization can be extended to analyze different MFA solutions.

Organization. Section 2 provides background on strong authentication solutions and SSO for native mobile apps, and on the formal specification language ASLan++ and the SATMC tool that we have employed. Despite the generality of our study, for concreteness, we develop our ideas in the context of two real use-case scenarios presented in Section 3. Having identified the scenario requirements, Section 4 describes the design of the proposed MFA reference models, identifies the corresponding security assumptions, discusses the peculiarities of a MFA solution compared to a basic username-password authentication, and identifies the corresponding threat model and security goals. The reference models are then formally specified (Section 5) and analyzed using SATMC (Section 6). Section 7 discusses related work and Section 8 draws conclusions. In Appendix A, we provide a glossary of abbreviations and notations used in the paper that the reader might find useful.

2 Background

This section provides the basic notions required to understand our reference models and their security assessment. In Section 2.1, we describe the roles involved in Multi-Factor Authentication (MFA) and Single Sign-On (SSO) solutions, discuss the different One-Time Password (OTP) generation approaches, and identify the functional requirements of a native SSO solution. In Section 2.2, we provide useful background for our formal analysis.

2.1 Multi-Factor Authentication and Native Single Sign-On

An authentication protocol determines the validity of one or more authentication factors used to claim a digital identity of an entity. In our analysis, we will focus on MFA solutions that augment the security of the basic username-password authentication by exploiting two or more authentication factors. In [15], it is defined as:

> *"a procedure based on the use of two or more of the following elements — categorized as knowledge, ownership and inherence: i) something only the user knows, e.g., static password, code, personal identification number; ii) something only the user possesses, e.g., token, smart card, mobile phone; iii) something the user is, e.g., biometric characteristic, such as a fingerprint. In addition, the elements selected must be mutually independent [...] at least one of the elements should be non-reusable and non-replicable".*

The more factors are used during the authentication process, the more confidence a service has that the user is correctly authenticated.

The following are the roles usually involved in an authentication solution in the context of native mobile apps:

- a *User* that wants to access a native *Service Provider app* (SP_{app}),
- an *Identity Provider server (IdP)* that manages the digital identities of the users and provides the MFA process, and
- a *User Agent (UA)*, which could be a browser or a native app used to perform the MFA process between SP_{app} and IdP.

Optionally, the SP_{app} could have a *backend server* (SP_S).

In the case of MFA, the authentication process usually involves also physical devices or software for the generation of additional identity proofs. We indicate with *Token Provider* (TP) the role that manages the generation and validation of additional identity proofs; TP may have a server side (TP_S) and a client side (TP_{app}). We indicate with *Hardware Token* ($HWToken$) a physical device used in combination with the user smartphone (e.g., smartcard or USB keys).

There are many MFA solutions on the market. In this paper, we focus on a well-accepted solution that combines a PIN code ("something only the user knows") with the generation of an OTP using an OTP generator ("something only the user possesses"). When an OTP-generation approach is used, a different password is generated for each authentication request and is valid only once, providing a fresh authentication property. Thus, compromising an old OTP does not have security consequences in the authentication process.

We can classify the many existing algorithms for generating OTPs into three main approaches:

- *Time-based OTP (TOTP)* [28]: a TOTP requires that the prover (TP_{app}) and verifier (TP_S) must either share the same secret or the knowledge of a secret transformation to generate a shared secret. The OTP is generated starting from the shared secret key (called *seed*) and the current time of the operation. TP_S must validate this value: only OTPs that fall into a short temporal range are accepted. TOTP is an extension of the *HMAC-based One-Time Password algorithm (HOTP)* [26]. HOTP computes the OTP using the seed plus a counter that is incremented every time an OTP is produced, whereas TOTP computes the OTP using the seed plus the current time.
- *Lamport's algorithm* [34]: the initial OTP is generated from a seed value and each successor OTP value is based on the value of its predecessor. For example, if s is a seed value and $F(x)$ is a one-way function, the following OTPs are generated: $o_1 = s, o_2 = F(o_1), o_3 = F(o_2), \dots o_n = F(o_{n-1})$. The last OTP, o_n, is stored on TP. When a *User* wants to login, she sends o_{n-1} to the server, and the server applies the function F and checks that the result is equal to the stored value. If this is the case, TP authenticates *User* and updates the stored value with o_{n-1}. In the next login, *User* will use o_{n-2} and so on. After n logins, *User* has to change the seed value and calculate new OTP values.

- *Challenge-Response (CR)* [27]: in the execution of this approach, the verifier (TP_S) presents a *challenge* (e.g., a pseudo-random number) and the prover (TP_{app}) answers with a valid *response* obtained by performing an established operation (such as hashing the challenge with a secret, or applying a private key operation). The response will be given to the verifier to check if it matches the expected value. Since the challenge is usually a fresh value, the response can be considered as a special case of OTP. In general, the use of a CR approach requires the interaction with an external *HWToken* (e.g., a smartcard or a smartkey) where a private key is securely stored.

In Section 4, based on our use-case scenarios, we will detail the TOTP and CR approaches.

Native SSO protocols allow users to access multiple SP_{app} through a single authentication performed with an *IdP*. As identified in [46], the two requirements that we expect for a native SSO solution are:

1. the *IdP* user credentials can be used to gain access to several SP_{app} — this implies that a *User* does not need to have credentials with a SP_{app} to access it;
2. if a *User* has already an authenticated session with an *IdP*, then she can access a new SP_{app} without re-entering her *IdP* credentials — only the consent of the *User* is required.

2.2 Formal Analysis: ASLan++ and SATMC

The use of formal languages and automatic tools for analyzing security protocols has allowed researchers to uncover a large number of vulnerabilities in protocols that had been thought to be, or even informally proved to be, secure. Famous examples range from protocols such as the Needham-Schroeder Public Key protocol to Kerberos or TLS (see [10] for details). These examples underline how the design of a protocol that requires specific security goals is not a simple task, as its security depends on several assumptions on trust and communication channels (e.g., the federation between the involved parties, and the transport protocol used in the message exchange). Several formal languages have been developed, all sharing the idea to extract from the protocol message flow a description of the entities involved, the exchanged messages and the channel assumptions. Formal protocol specifications are then given as input to automated tools (e.g., Tamarin [36] and ProVerif [12]) that check the desired security goals of the protocol against realistic threat models.

In our analysis, given our expertise and past experience, we use ASLan++ [8, 53] and SATMC [4]. ASLan++, the input specification language of the AVANTSSAR Platform [3], is a high-level formal language that formalizes the interactions between the different protocol roles, where a role represents a sequence of operations (e.g., sending and receiving messages) that must be executed by the entity that plays that role. ASLan++ supports the specification of different channel assumptions and security goals, most notably different variants of authentication and confidentiality. SATMC, which is one of the model checkers of the AVANTSSAR platform, uses state-of-the-art SAT solvers and allows for the specification of security goals written using the Linear Temporal Logic.

3 Use-Case Description

Despite the generality of our research, for concreteness, we consider two real use-case scenarios:

1. *TreC* (Section 3.1), which involves mobile health (mHealth) apps and the use of a Time-based OTP (TOTP) approach, and
2. *DigiMat-Lab* (Section 3.2), which involves the Italian electronic identity smartcard and a Challenge-Response (CR) approach.

Both scenarios need the design of a MFA solution with a SSO experience for mobile native apps. As we explained in Section 1, the use of current browser-based authentication protocols in the mobile context requires a detailed understanding of the security issues as well as the design of a flow specific to mobile native

Table 1: Requirements for the two real-world use-case scenarios.

Project	Scenario	Requirements
TreC	mHealth	1) TOTP approach
		2) Use of the existing mobile native app for supporting MFA and SSO in the mobile context
DigiMat-Lab	eID scheme	1) CR approach with the use of the Italian electronic identity card (CIE 3.0)
		2) Support of SAML as browser-based authentication protocol

apps. Some guidelines for supporting authentication and authorization in the mobile context were released in 2017 by the OAuth working group, namely "OAuth 2.0 for Native Apps" [40]. Considering our scenarios, the problem is that this specification does not mention how to extend the protocol to support MFA solutions or other IdPs that support different authentication protocols.

Given these limitations, Section 4 describes the two reference models that we have designed based on the requirements described in the following subsections, and summarized in Table 1, for the two real-world use-case scenarios that we consider.

3.1 TreC: mHealth

TreC (an acronym in Italian for "Cartella Clinica del Cittadino", i.e., "Citizens' Clinical Record") is a platform developed in the Trentino region (in the north-east of Italy) for managing personal health records (PHRs) [42]. Besides for the web platform[3], which is routinely used by around 98,000 users, TreC has been implementing a number of native Android apps to support self-management and remote monitoring of chronic conditions. These apps are used in a "living lab" by voluntary chronic patients according to their hospital physicians. Examples are:

- *TreC Diario Diabete*, a mobile diary that allows patients to record health data, such as the blood glucose level and physical activity, and
- *TreC FSE*, a mobile native app that permits patients to consult their personal health data and medical prescriptions from the smartphone.

To provide a MFA solution for the web scenario, TreC developers have released a mobile native app that is used as an OTP generator: after the entering of a PIN value, the app shows an OTP on the screen of the mobile device that is then copied by the user on the web page together with the credentials. To take advantage of the presence of the application that users must download to authenticate themselves in the web scenario, a requirement of the TreC scenario was to extend the existing application to support the native SSO as well.

Fig. 1 shows two screenshots of the beta version of the *TreC FSE* app, which we plan to extend to support SSO and MFA for the TreC app ecosystem.

3.2 DigiMat-Lab: the Italian Electronic Identity Card

DigiMat-Lab is a joint laboratory between FBK and IPZS, which is the Italian state printing office and mint.[4] In the context of the laboratory, the FBK team have been involved in various activities, most notably in the design of a mobile MFA mechanism that uses the Italian electronic identity card (CIE 3.0 – Carta d'Identità Elettronica) [37] as an OTP generator. CIE 3.0 is a personal identification document that will replace the

[3] `https://trec.trentinosalute.net/`

[4] The acronym IPZS stands for "Istituto Poligrafico e Zecca dello Stato". More information is available at `https://www.ipzs.it/`.

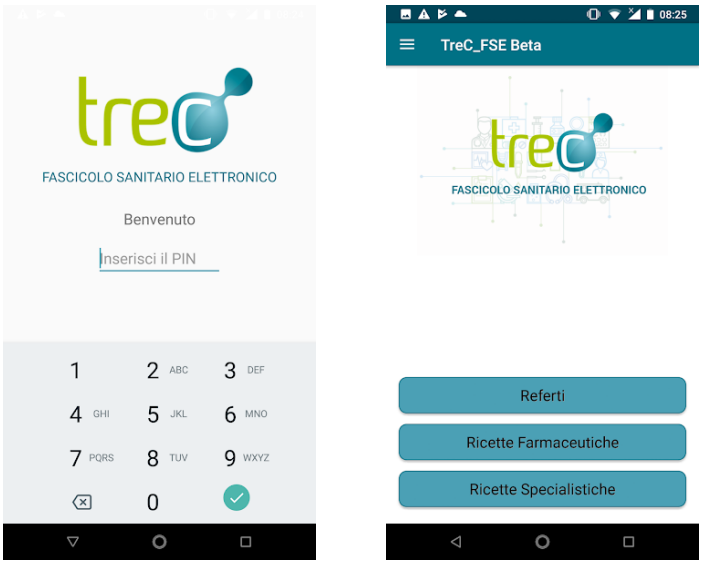


Figure 1: Screenshots of the TreC App.

paper-based identity card in Italy and is used for both online and offline identification. The microprocessor of CIE 3.0 is contactless and can be read using a smartphone with a NFC (Near Field Communication) interface.

We have derived two requirements for the DigiMat-Lab scenario. The choice of the OTP approach is based on the CIE 3.0 characteristics. This card carries a private key and a X.509 certificate issued by the Certification Authority of the Italian Ministry of Internal Affairs with the corresponding public key. Encryption, decryption and signature with this certificate are the basic functionalities of this card, which is therefore suitable for a CR OTP approach (cf. Section 2.1). The *UA* and protocol choice is based on the identity provider characteristics. In Italy, to be in line with the majority of identity providers of the public administration, we need a solution that supports SAML SSO [39], which is a widely-used browser-based authentication protocol.

Fig. 2 shows two screenshots of the CieID mobile app that was developed to manage authentication by interacting with the CIE 3.0. The current solution on the market, which has been certified with eIDAS [17], supports only mobile web applications. The presented reference model is an extension of it to support mobile native applications.

4 Description of our Reference Models

In this section, we present the two mobile MFA and SSO reference models that we have derived from the use-case scenarios described in Section 3: RM_{TOTP} and RM_{CR}, where a reference model is an abstract (semi-formal) representation of an authentication flow. In both reference models, we focus on scenarios where the server-side of the Identity Provider (*IdP*) and Token Provider (TP_S) are played by the same entity that we called *IdTP*. The entity-role mappings between the roles of an *Identity Management* (*IdM*) system (cf. Section 2.1) and the entities of our reference models are summarized in Table 2 (our reference models can be used in different scenarios, thus we do not specify a specific service provider app SP_{app}). In detail:

RM_{TOTP}. We have designed this reference model starting from the TreC use-case requirements. The OTP generator app is released by *IdTP* and plays both the role of the User Agent (*UA*) and the Token

Figure 2: Screenshots of the CieID App.

Provider client-side (TP_{app}). Abstracting from the TreC scenario, we call this app *IDOTP* (see Table 2).

To design this reference model, as described in [46], we have first performed a rational reconstruction of the native SSO used in Facebook (FB) [18]. To obtain a precise description of the FB native SSO flow, we have used the following methodology: we have analyzed the source code of the FB SDK to understand the interaction between a client app and the FB client application inside the smartphone, and we have used the Fiddler proxy tool[5] to carefully inspect the HTTP(S) traffic between the FB client and the FB server. We have derived a reference model with a native app as *UA* that generalizes the one proposed by FB in such a way that it can be used as a reference model by any IdP willing to provide its own *IdM* solution. In [46], we also made a comparison between the derived reference model and OAuth for native apps [40]. In this work, compared to the reference model proposed in [46], we have enhanced the security by adding the generation, exchange and validation of Time-based OTPs (TOTPs). The MFA based on TOTP protects mainly against a stolen smartphone. Indeed, even if the user's smartphone is stolen, the attacker cannot login as the victim without generating the expected OTP. We have designed this reference model by taking into account a security-by-design paradigm (i.e., the solution has been designed from the first steps to be secure) and usability aspects (i.e., we have followed a human-centered design approach that has the goal of making usable solutions, having in mind the user's needs and requirements); our goal was to propose a solution that guarantees a high level of security while maximizing usability. For example, RM_{TOTP} does not require users to digit (long) OTP strings as they are directly communicated by *IDOTP* (the TOTP generator app) to the *IdTP*. This obviously improves the user experience, and prevents the theft of OTP values from the clipboard [19], which could be used by *User* to copy the OTP value if manually entered.

RM_{CR}**.** We have designed this reference model starting from the *DigiMat-Lab* use-case requirements. As SAML 2.0 was not designed taking into account the mobile scenario, we have extracted from the OAuth for native apps [40] all the security features that can be adopted in SAML. Like in [40], our reference model requires the use of an external user agent and does not save client's secret on the app. The main difference is the flow choice. In [40] the suggested flow is *Authorization Code flow with PKCE* [30]. In our reference model, since the OAuth flow is incompatible with SAML IdPs, we had to adapt the

[5]Available at `http://fiddler2.com/`.

Table 2: Mapping our Reference Model Entities and *IdM* Roles.

IdM Role	RM_{TOTP}	RM_{CR}
User Agent (UA)	*IDOTP*	*Browser*
Service Provider app (SP_{app})	SP_{app}	SP_{app}
Token Provider app (TP_{app})	*IDOTP*	*OTPApp*
Identity Provider (IdP) and *Token Provider* (TP_S)	*IdTP*	*IdTP*
Hardware Token (HWToken)	-	*eIDCard*

suggested flow.[6] To mitigate token interception attacks, thus guaranteeing similar security properties of PKCE, we require the use of App Link as a redirection method (instead, it is optional in [40]). Hence, we have generalized [40] by providing a reference model that can be used by SAML-based IdP and for supporting a MFA based on the generation of OTPs with a CR approach. As explained in Section 2, the use of a CR approach requires the interaction with an external *Hardware Token* (*HWToken*), where a private key is stored securely. Focusing on the *DigiMat-Lab* project, we describe the reference model for electronic identity cards (*eIDCard*) protected by a PIN that interact with a native app (called *OTPApp*) that plays the role of TP_{app} (see Table 2).

In general, we could consider different combinations of OTP generator approaches or scenarios where the second-factor validation is performed by *IdP* and TP_S belonging to different security domains (these changes require extra design and we will consider them in future work; in any case, for certain combinations, we can simply reuse our existing building-blocks).

Both RM_{TOTP} and RM_{CR} consist of three phases: *registration*, *activation* and *exploitation*, which we describe in the following subsections. As the registration and activation phases are two preliminary phases, and our focus is on the authentication property performed during the exploitation phase, we provide more details for the exploitation by describing the reference model flow with a step-by-step description using message sequence charts and detailing the required security assumptions.

4.1 Registration Phase of RM_{TOTP} and RM_{CR}

The registration phase is performed by the SP_{app} developers and corresponds to the exchange of some information (called *metadata*) about SP_{app} with *IdTP*. The metadata required by *IdTP* can vary, but at least the following two values must be always present (see Table 3):

RM_{TOTP}. The app package name and the certificate fingerprint (called *key_hash*) of the app must always be present. The *key_hash* is a digest (SHA1) of the file CERT.RSA, which contains the public key of the developer, the signature of the app package calculated using the private key of the developer, and other information about the certificate. *key_hash* depends on the private key of the SP_{app} developer and it is thus different for apps implemented by different developers. After the registration phase, the *IdTP* associates an identifier with the registered SP_{app}.

RM_{CR}. The app package name and the redirection URI (called *redirect_uri*) of the app must always be present. A *redirect_uri* corresponds to the HTTPS scheme URI claimed by SP_{app} to manage the redirection inside the smartphone. As explained in [40], "some platforms allow apps to claim an HTTPS scheme URI after proving ownership of the domain name. URIs claimed in such a way are then opened in the app instead of the browser." In Android (6.0+)[7] this mechanism is called *App Links* [1] (whereas in iOS (9+) it is called *Universal Links* [32]). To register an HTTPS URI scheme the developer has to perform the following steps: configure the app to support HTTPS URI scheme,

[6] If the IdP is OIDC compliant, then we suggest to refer to the current best practice [40, 31].

[7] For previous Android versions, the app-declared custom URI scheme can be used as an alternative redirection scheme. A known limitation of this scheme is that the registration of a specific URI is not unique. See [40] for more details.

and publish on the developer's website a JSON file with the URLs that the app manages. This file is used to provide domain verification.

The registration phase can be performed in different ways, e.g., entering the data into an online dashboard or via an email exchange. Since a trust relationship between SP_{app} and *IdTP* is established as result of the registration phase, it is important that *IdTP* validates the SP_{app} data; in some cases (e.g., when user personal or sensitive data are involved) a service-level agreement could be required as well.

4.2 Activation Phase of RM_{TOTP} and RM_{CR}

The activation phase is performed by the *User* to configure TP_{app} to support a second authentication factor as well:

RM_{TOTP}. As explained in Section 2, a TOTP algorithm requires that "the prover and verifier must either share the same secret or the knowledge of a secret transformation to generate a shared secret" [28], without specifying when and how to exchange this secret. For RM_{TOTP}, *IDOTP* obtains from *IdTP* the seed value (i.e., a shared secret between *IDOTP* and *IdTP* used, combined with the time, to generate new OTP values) as part of the activation phase, and then stores it encrypted with the *PIN* code selected by the *User* during this phase (we indicate this with $\{|seed|\}\_PIN$, where the notation $\{|M|\}\_K$ means that a message M is encrypted with a symmetric key K).[8] In addition, as a consequence of the activation, *IDOTP* obtains from *IdTP* a token called *token_IdP* (used as a session token in place of the user credentials to provide a SSO experience).

RM_{CR}. As a consequence of the activation, *OTPApp* asks *User* to digit the PIN code of her *eIDCard* (a value known to *User*) and to put in contact the *eIDCard* with her smartphone. In the interaction between the *eIDCard* and *OTPApp* the PIN value is checked. At the end of this step, *OTPApp* stores internally the second part of the PIN code (called *PIN2*). We have decided to store the second part of the PIN code and the *User* will be required to insert only the remaining four digits from here on for both usability (usually an *eIDCard* PIN is composed of 8-digits) and security (*PIN2* becomes an additional ownership factor that an attacker has to compromise) reasons.

Since in our analysis we will focus on the authentication property, we assume that the registration and activation phases have been already performed, without any interference of attackers. This is a common practice [33, 44] when dealing with authentication protocols consisting of a preliminary phase in which specific values are stored to set the protocol up. Indeed, since a trust relationship between SP_{app} and *IdTP* is established as a result of the registration phase, it is important that *IdTP* validates the SP_{app} data; if the activation phase is compromised (e.g., by a phishing app activated in place of the valid one), then no security guarantees can be achieved during the authentication as the attacker is able to generate valid OTPs and use them during the exploitation phase.

Table 3 summarizes the values stored by the involved entities as a result of the registration and activation phase. These values represent the initial knowledge of the entities before the execution phase and are shown in Fig. 3 and Fig. 4 inside gray boxes.

4.3 Exploitation Phase of RM_{TOTP} and RM_{CR}

The exploitation phase is performed every time the *User* accesses a SP_{app}. The steps for RM_{TOTP} are shown in Fig. 3 and consist of:

S1. *User* opens SP_{app}.

S2. SP_{app} sends a request to SP_S including a session token *token_sync*.

[8] As a common assumption is to consider cryptography to be unbreakable, we are simply modeling the seed stored encrypted with the PIN. In a real implementation, to avoid brute force attacks, the seed should be encrypted with a longer key derived from the PIN value or the server should set a limit on the wrong login requests (i.e., requests with a not valid OTP value).

Table 3: Registration and Activation Phases.

	Action	RM_{TOTP}	RM_{CR}
Registration Phase	Exchange of *SP* metadata	package name, *key_hash*	package name, *redirect_uri*
Activation Phase	Values stored in *UA* for SSO	*token_IdP*	-
	Values stored in TP_{app}	$\{\|seed\|\}\_PIN$	*PIN2*
	User knowledge	*PIN*	*PIN1*

S3. SP_S checks the validity of *token_sync*. If *token_sync* has expired, then SP_S sends an error message asking for a login to SP_{app}, otherwise Step S7 is executed.

A1. *User* clicks the login button.

A2. SP_{app} sends a token request to *IDOTP* passing as parameter its identifier.

A3-A4. *IDOTP* reads the *key_hash* value of SP_{app}, and sends a token request to *IdTP* including this value. *key_hash* is used by *IdTP* to validate the SP_{app} identity.

A5. If SP_{app} is valid, then *IdTP* returns to *IDOTP* a consent containing the meta-data of SP_{app}.

A6. *IDOTP* asks *User* the *PIN* value showing the SP_{app} information.

A7. If *User* agrees to the operation shown by *IDOTP*, then *User* enters the *PIN* value.

A8-A9. *IDOTP* generates an OTP as a function of the seed (extracted using the *PIN* as decryption key) and of time, and sends it to *IdTP* together with SP_{app}, the *key_hash* value and *token_IdP*, which corresponds to the user credentials entered during the activation phase.

A10. If the OTP value is valid, then *IdTP* returns a token token_SP=$\{IdTP, User, SP_{app}\}_{K^{-1}_{IdTP}}$ to *IDOTP*. token_SP contains the identity of *IdTP*, *User* and SP_{app}, and is digitally signed with K^{-1}_{IdTP}, the private key of *IdTP*. In general, the notation $\{M\}_{K_A^{-1}}$ means that the message M is digitally signed by A using its private key K_A^{-1}.

A11. *IDOTP* returns token_SP to SP_{app} as result of Step A2.

S4. SP_{app} sends a token request to SP_S including token_SP.

S5. SP_S checks the validity of token_SP. To perform this validation, SP_S should check that token_SP: *(i)* is not expired or is received within a limited amount of time (e.g., 5 minutes from the token request); *(ii)* contains the right SP_{app} identifier; and *(iii)* is released by the right *IdTP* (by checking the token signature and *IdTP* identifier). If token_SP is valid, then SP_S creates and sends to SP_{app} a token *token_sync*. This token will be used by SP_{app} to synchronize user data in future interactions, until its expiration.

S6. When SP_{app} needs to synchronize data, SP_{app} sends a request to SP_S including *token_sync*.

S7. SP_S checks *token_sync* and, if *token_sync* is valid, then SP_S sends the requested resource to SP_{app}.

We have labeled the steps with "S" and "A". The S steps are related to the *SP* (but note that our representation is only an example and each *SP* could support different solutions). The A steps represent the steps related to authentication. As the S steps can vary depending on the choices of the SP developers, in our analysis, we will focus on the A steps.

The A steps for RM_{CR} are shown in Fig. 4:

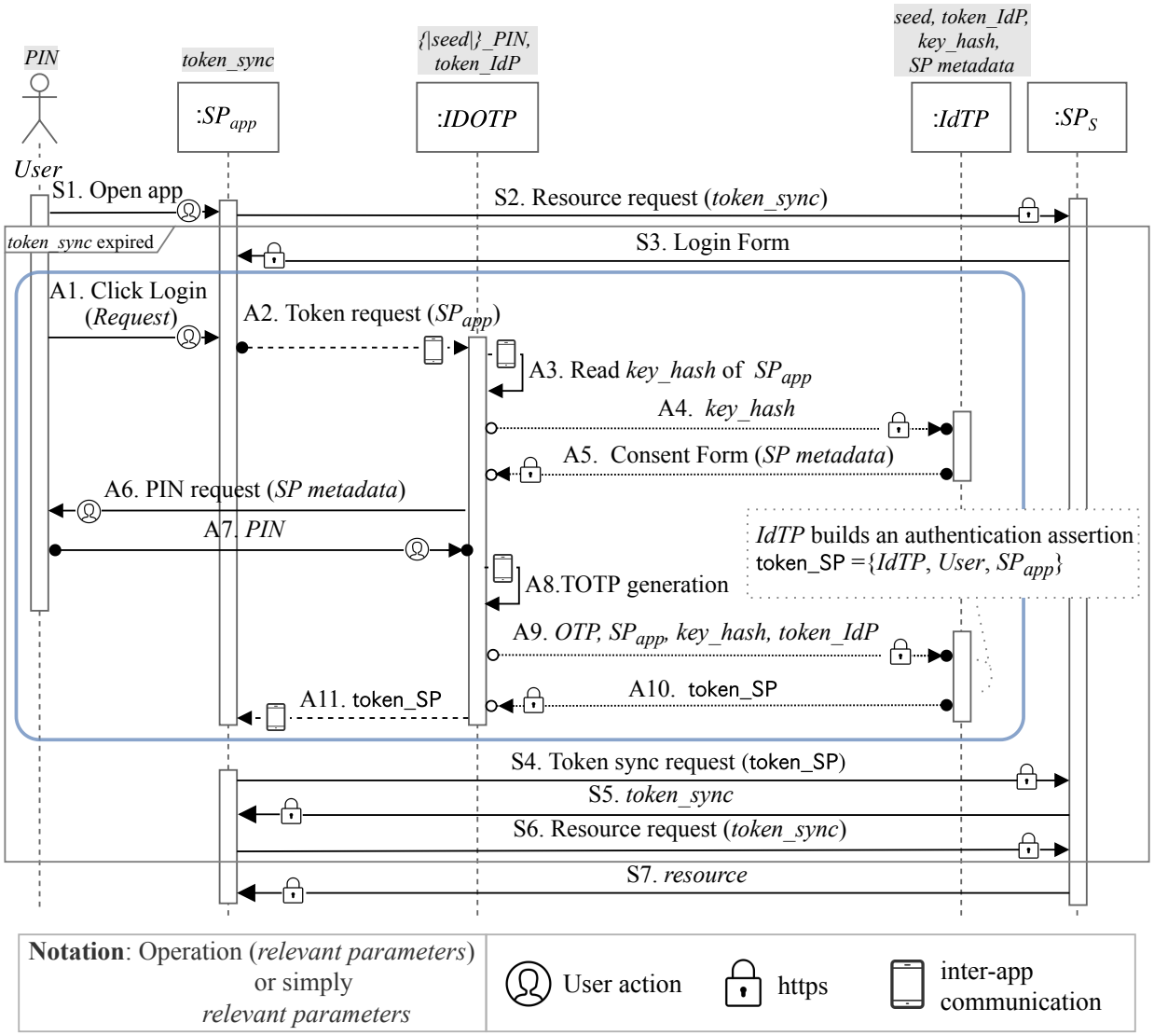


Figure 3: RM_{TOTP} Flow.

A1. *User* clicks the login button.

A2. SP_{app} opens *Browser* with a token request to *IdTP* passing as parameter its identifier.

A3. *Browser* follows the link redirection and sends the token request to *IdTP*.

A4-A5. *IdTP* returns a challenge to *OTPApp*, passing through *Browser*, where a cookie (called *session_cookie*) is set.

A6-A8. *OTPApp* asks *User* to digit the first part of her PIN (called *PIN1*) specifying the operation (in this case the login with SP_{app}) and to place her *eIDCard* close the her smartphone.

A9. *OTPApp* combines the two parts *PIN1* (value just typed) and *PIN2* (value stored during the activation phase) of the PIN, and starts the communication with the *eIDCard* via NFC sending the combined PIN and the challenge.

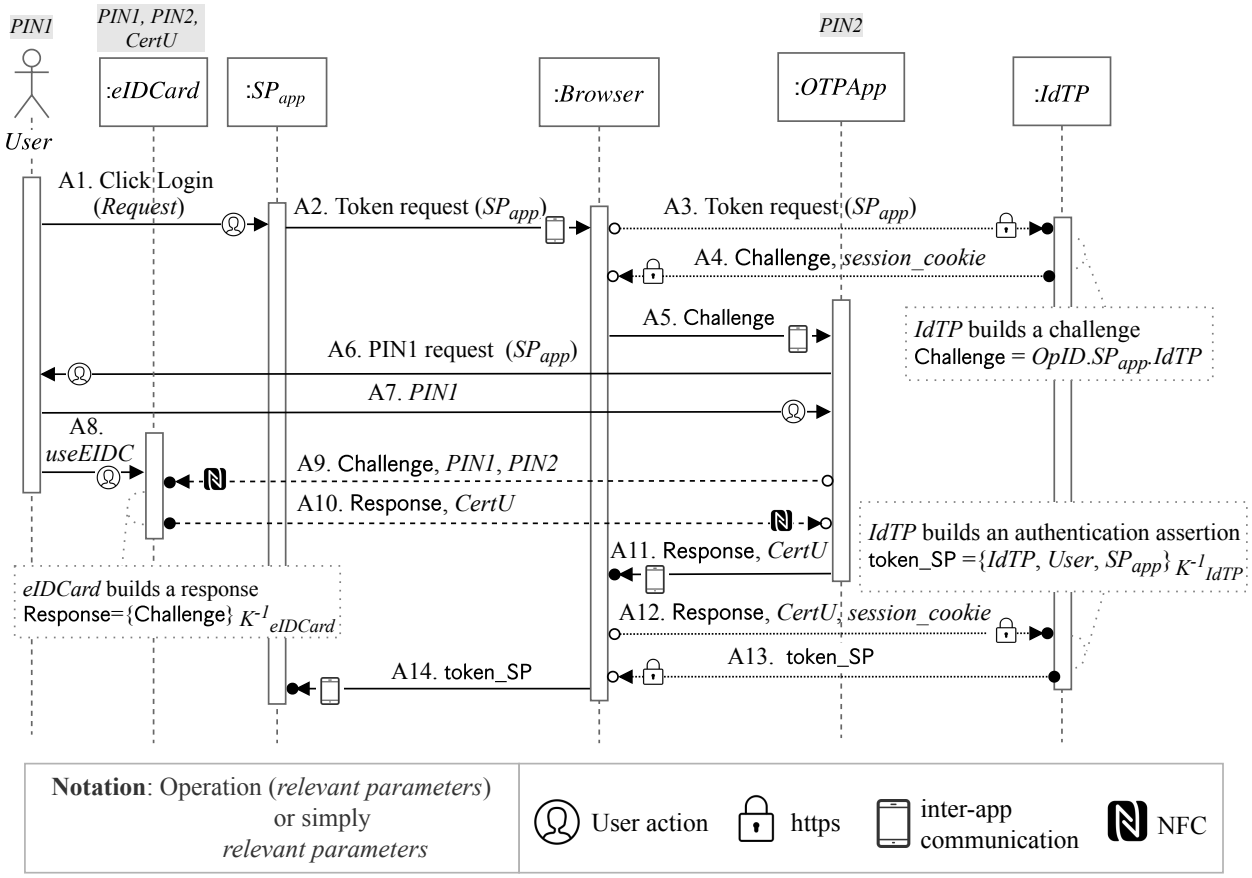


Figure 4: RM_{CR} Flow.

A10. After checking the PIN value, *eIDCard* signs the challenge using the private key stored inside the chip and sends the generated response together with the user X.509 certificate (called *CertU*) to *OTPApp*.

A11-A12. *OTPApp* sends, through *Browser*, the response value and *CertU* to *IdTP*. In Step A12, *Browser* adds the corresponding *session_cookie*.

A13. If the response is valid, then *IdTP* returns to *Browser* a token token_SP that contains the identity of *User*, *IdTP* and SP_{app}, and is digitally signed with K^{-1}_{IdTP}, the private key of *IdTP*.

A14. *Browser* redirects token_SP to SP_{app} following its *redirect_uri* (value stored by *IdTP* during the SP_{app} registration phase).

The challenge value can be selected by the *IdTP*. In our analysis, we consider the following concatenation of messages: challenge = $OpID.SP_{app}.IdTP$, where *OpID* is the identifier of the authentication request.

4.4 Security Assumptions

Our reference models are based on a number of security assumptions (sa) divided in:

- *Trust Assumptions*, which clarify the trust relationships between the different entities,
- *Communication Assumptions*, which specify the concrete implementation of the communication channels required in our reference models, and

- *Activation Assumption*, which identifies the assumptions related to the activation phase.

We use a subscript ($TOTP$ or CR) to specify when the assumption is specific for a reference model. When the subscript is not specified, the assumption holds for both the reference models.

4.4.1 Trust Assumptions.

For both RM_{TOTP} and RM_{CR}, we have identified the following trust assumption:

(TA1) *IdTP* is trusted by SP_{app} on identity assertions, i.e., *IdTP* releases only valid identity assertions.

In addition, for RM_{CR} we have:

(TA2$_{CR}$) *Browser* is trusted by *User* and *IdTP* on receiving OTP values and identity assertions.

4.4.2 Communication Assumptions.

For RM_{TOTP}, communications between the entities are subject to the following assumptions:

(CA1$_{TOTP}$) The communication between SP_{app} and *IDOTP* is carried over an inter-app communication implemented using `StartActivityForResult()`. This Android method allows an app to execute another app and get a result back, and thus guarantees that the SP_{app} that sends a request to *IDOTP* at Step A2 in Fig. 3 is the same app that receives the result back from *IDOTP* at Step A11.

(CA2$_{TOTP}$) To read the *key_hash* value (Step A3 of Fig. 3), *IDOTP* uses the Android method `getPackageInfo(packageName, PackageManager.GET_SIGNATURES)`, which extracts the information about the certificate fingerprint included in the package of SP_{app}.

(CA3$_{TOTP}$) The communication between *IDOTP* and *IdTP* occurs over a unilateral TLS channel, established through the exchange of a valid certificate (from *IdTP* to *IDOTP*).

For RM_{CR}, communications between the parties are subject to the following assumptions:

(CA1$_{CR}$) To exchange token_SP with SP_{app} (Step A14 of Fig. 4), *Browser* uses Android App Links. This redirection scheme guarantees that *Browser* sends token_SP to the correct SP_{app} (previously validated by the operating system as explained in Section 4.1).

(CA2$_{CR}$) The communication between *Browser* and *IdTP* occurs over a unilateral TLS channel, established through the exchange of a valid certificate (from *IdTP* to *Browser*).

(CA3$_{CR}$) *OTPApp* launches an external *Browser*. To increase usability, we suggest to use the Chrome Custom Tabs ("a programmatic instantiation of the browser that is displayed inside a host app, but retains the full security properties and authentication state of the browser" [40]).

(CA4$_{CR}$) The communication between *eIDCard* and *OTPApp* occurs over a unilateral secure channel, established through the exchange of a valid certificate from *eIDCard* to *OTPApp*.

In order to show that these assumptions are reasonable, they refer to a concrete implementation of the communication channels. However, our analysis is general and in Section 5.3 we will provide the formal counterpart abstracting away the implementation details. By doing so, any implementation satisfying the abstract assumptions can be used in place of the implementation mentioned above (e.g., considering a similar solution in the case of iOS), and the results of our security analysis still hold.

4.4.3 Activation Assumption.

Phishing attacks (in this context, we mean a malicious app that creates a fake login form and steals the user's credentials) are one of the most common types of attack and usually are beyond the scope of an authentication protocol. In our analysis, together with a secure communication, we assume that no phishing is possible during the activation phase of RM_{TOTP} and RM_{CR} (while we will consider it during the exploitation phase analysis):

(ActivA) The activation phase is correctly performed by *User*. That is, *User* downloads the correct *IDOTP* or *OTPApp* (i.e., they are not fake apps) and correctly follows the activation phase, and the communication channels that are involved in this phase are secure.

4.5 Towards a Formal Specification of Multi-Factor Authentication

Let us now discuss the peculiarities of a MFA solution compared to a basic username-password authentication; in doing so, we introduce some concepts that will be key for the formal analysis.

A MFA solution augments the security of the basic username-password authentication by exploiting two or more authentication factors. Following the definition given in Section 2.1, we infer that RM_{TOTP} and RM_{CR} are two-factor authentication solutions using knowledge and ownership elements (factors). As the real-world scenarios that we have described in Section 3 do not involve inherence elements, we leave their analysis as future work. To assess in a finer-grained way the security of solutions where different factors belonging to the same category are used (e.g., a solution that requires a PIN and a password), we introduce the notion of instance factors.

Given a protocol P, we call *instance factor* (IF_P) every specific instance of either an ownership factor ($IFactor_o$) or a knowledge factor ($IFactor_k$) involved in P.

As shown in Table 4, RM_{TOTP} involves the following three instance factors:

- the $IFactor_o$ *token_IdP* that is stored in *IDOTP* and in *IdTP* as a result of the activation phase (used as a session token in place of the *User* credentials to provide a SSO experience),
- the $IFactor_k$ *PIN* known by the *User* (used to protect the OTP generator), and
- the $IFactor_o$ $\{|seed|\}\_PIN$, a value stored in *IDOTP* and shared (without the *PIN* encryption) with *IdTP*.

RM_{CR} involves the following three instance factors:

- the $IFactor_o$ *eIDCard* that is owned by the *User* and contains her private key used to sign the challenge request,
- the $IFactor_k$ *PIN1* that is known by the *User* and entered in *OTPApp* during the exploitation phase, and
- the $IFactor_o$ *PIN2* that is stored in *OTPApp* and that, combined with *PIN1*, is used to protect the *eIDCard*.[9]

Compared to the usual notion of authentication factors [15], instance factors can have a dependency. For example, the two ownership instance factors of RM_{TOTP} are both stored in *IDOTP*. Thus, by breaching the *IDOTP* app both of them are compromised. However, it is important to note that in general different mitigations can be implemented for the different instance factors, thus considering instance factors provides a finer-grained security analysis. For example, in our reference model, if a *User* realizes that the *IDOTP* has been compromised (e.g., if her smartphone has been stolen), she can invalidate *token_IdP*, thus blocking possible attacks.

[9] We consider *PIN2* as an ownership factor as we assume that users are educated to digit the entire *PIN* only during the activation phase, and thus users will never insert *PIN2* anymore.

Table 4: Instance Factors.

	$IFactor_k$	$IFactor_o$
$IF_{RM_{TOTP}}$	*PIN*	*token_IdP*, $\{\|seed\|\}$*_PIN*
$IF_{RM_{CR}}$	*PIN1*	*eIDCard*, *PIN2*

Table 5: Attackers of our Threat Model.

Attacker Name	**Description**
Smartphone Device Thief ($DT_{\square}$)	a proximity attacker who is able to steal the *User*'s smartphone
eIDCard Device Thief ($DT_{\blacksquare}$)	a proximity attacker who is able to steal the *User*'s electronic identity smartcard
Social Engineer (*SE*)	a hacker attacker capable of fooling or persuading the *User* into handing over confidential or sensitive data (e.g., the user's password or PIN), using for example a fake email
Shoulder Surfer (*SS*)	a proximity attacker who is able to read the data entered by the *User* during the authentication
App Duplicator (AD)	a proximity-hacker attacker who is able to clone the instance factors stored in the smartphone
Eavesdropping Software (*ES*)	a malware inside the *User*'s smartphone able to read the data entered by the *User* (e.g., keylogger)
Malicious Application (*MA*)	a malicious application inside the *User*'s smartphone that is able to interact with the *User* and perform phishing attacks

4.6 Threat Model

Based on our use-case scenarios (cf. Section 3) and in relation to the capabilities that an attacker can have under the security assumptions of Section 4.4, we have identified a number of proximity and hacker attackers (inspired by the NIST threat model in [24] and [49]), which we list in Table 5. We model these attackers taking into account two aspects:

- specifying how instance factors can be explicitly compromised (*operations* performed), and
- detailing which instance factors are explicitly compromised by the attacker (*set of instance factors*).

4.6.1 Operations of Attackers.

As mentioned in Section 4.5, we have considered two kinds of instance factors: $IFactor_o$, consisting of a value stored by an entity (e.g., a seed value stored on an OTP generator app), and $IFactor_k$, which is a value known by the *User* and entered into one or more entities (e.g., a PIN value entered into an OTP generator app). Then, we can distinguish four kinds of operations that an attacker in our threat model can perform:

know($IFactor_k$): an attacker could know the $IFactor_k$ value independently of the protocol run (e.g., by persuading the *User* to reveal the PIN value).

own($IFactor_o$): an attacker could have the ability to extract $IFactor_o$ from an entity (e.g., by reading the app data).

overhear(E): an attacker could have the ability to overhear messages sent by *User* to the entity E.

interact(E): an attacker could have the ability to interact with the entity E in place of an honest entity (e.g., the *User*). If the interaction of an entity with another is mutually exclusive, namely $interact_{ME}(E)$,

Table 6: Attacker Operations and Compromised Instance Factors for RM_{TOTP} and RM_{CR}.

Attacker (a)	**Operation**	$IF_{RM_{TOTP}}(a)$	$IF_{RM_{CR}}(a)$
$DT_{\square}$	for all E in $\square$. $interact_{ME}(E)$	$\{\|seed\|\}\_PIN$, $token\_IdP$	$PIN2$
$DT_{\blacksquare}$	$interact_{ME}(\blacksquare)$	-	$eIDCard$
SE, SS	for all $IFactor_k$ entered by $User$. $know(IFactor_k)$	PIN	$PIN1$
AD	for all E in $\square$. $own(IFactor_o)$	$\{\|seed\|\}\_PIN$, $token\_IdP$	$PIN2$
ES	for all E in $\square$. $overhear(E)$	PIN	$PIN1$
MA	for all E in $\square$. $overhear(E)$,	PIN	$PIN1$
	$interact(\blacksquare)$,	-	$eIDCard$
	$interact(E)$, $own(IFactor_o)$	$\{\|seed\|\}\_PIN$, $token\_IdP$	$PIN2$

either the *User* or the attacker can interact with E. We require the mutual exclusion in case of stealing an entity (e.g., an attacker cannot ask the *User* to enter her PIN value into an app installed on a stolen smartphone).

The first two operations correspond to an attacker who is directly compromising (knows or owns) *IFactor* values. The last two operations correspond to an attacker who during the execution of the protocol intercepts a value expected by E or interacts with E (i.e., an attacker who is *compromising an entity*).

Table 6 shows how the attackers of Table 5 are modeled using these four operations. In detail:

- $DT_{\square}$ The theft of the *User*'s smartphone implies the capability to interact in a mutual exclusive way with all the entities that store an $IFactor_o$ in the smartphone. Thus, we model this attack as $interact_{ME}(E)$ for all E in $\square$.
- $DT_{\blacksquare}$ To model the possession of a smartcard, we consider an attacker able to interact with it in a mutual exclusive way. Thus, we model this attack as $interact_{ME}(\blacksquare)$.
- SE, SS SE and SS are able to know the $IFactor_k$ entered by *User* before the protocol run, thus we use $know(IFactor_k)$ for all $IFactor_k$ known by *User* and entered by the *User* during the authentication.
- AD To model an AD attacker, which is a proximity hacker with the ability to extract data from the device, we use $own(IFactor_o)$ for all E in $\square$.
- ES To model ES (e.g., a keylogger capable of reading all the data entered by the *User*), we use $overhear(E)$ for all E in $\square$.
- MA It could be the case that both ownership and knowledge instance factors are compromised. Indeed, if MA is a malicious app installed in the *User*'s smartphone, then it can persuade the *User* to enter the $IFactor_k$ (modeled as $overhear(E)$ for all E in $\square$), interact with the smartcard, thus compromising an $IFactor_o$ (modeled as $interact(\blacksquare)$), interact with all the entities that store an $IFactor_o$ in the smartphone (modeled as $interact(E)$ for all E in $\square$), and finally, with root privilege permissions, it is able to read the internal state of another entity (modeled as $own(IFactor_o)$ for all E in $\square$).

In our analysis, we have considered the attackers that are the most significant to our study. However, as remarked above, our approach is still applicable in case an analyst wishes to consider other attackers by combining $own(IFactor_o)$, $interact(E)$, $interact_{ME}(E)$, $know(IFactor_k)$ and $overhear(E)$ in different ways.

4.6.2 Set of Explicitly Compromised Instance Factors.

Together with the specification of how instance factors are compromised (the *operation* performed), we specify which instance factors are explicitly compromised by an attacker. Given an attacker a and a protocol P, we refer to this set as the *set of instance factors explicitly compromised by a in P* and denote it by writing $IF_P(a)$. Explicitly compromised means that for sure an attacker, given his capabilities, is able to

compromise an instance factor. However, based on the protocol, an attacker could also have the possibility to implicitly exploit other instance factors. For example, let us consider a MFA protocol based on passwords and an out-of-band (OOB) software [24]. An attacker who is able to explicitly compromise a password (e.g., a shoulder surfer) can start a run of the protocol on his computer. Then, if the user accepts the notified operation on his OOB software, the attack succeeds. In this case, an attacker is thus able to *(implicitly) compromise* the OOB software.

The relations between the attacker operations and $IF_P(a)$ are straightforward. $IF_P(a)$ is the smallest set such that:

- if $know(IFactor_k)$, then $IFactor_k \in IF_P(a)$;
- if $own(IFactor_o)$, then $IFactor_o \in IF_P(a)$;
- if $overhear(E)$, then $IF_P(a) \supseteq \{IFactor_k \mid IFactor_k \in E\}$;
- if $interact(E)$ or $interact_{ME}(E)$, then $IF_P(a) \supseteq \{IFactor_o \mid IFactor_o \in E\}$.

For example, if an attacker a can read all the information typed by *User* by performing $overhear(E)$, and in P *User* has to enter into E her *password* and *PIN*, then $IF_P(a) = \{password, PIN\}$.

Columns 3 and 4 of Table 6 show the set of instance factors associated with the aforementioned attackers in our reference models, where E=*IDOTP* in RM_{TOTP} and E=*OTPApp* or E=*eIDCard* in RM_{CR}. This definition can be easily extended to a set of attackers A as follows: $IF_P(A) = \bigcup_{a_k \in A} IF_P(a_k)$.

4.7 Definition of Goals G and G_n

In the context of SSO, the expected security goal of a basic password-based authentication is:

(G) *SP* authenticates *User* through an *IdP* assertion.

For G to hold, *User* is required to prove the possession and control of an authentication factor: either credentials (something only she knows) or a session token (e.g., a cookie stored in her browser) in order to be properly identified by *IdP* and consequently by *SP*.

We are not aware of any formal definition of the security properties of a MFA solution apart from [6]. In [6], the authors analyzed a two-factor and two-channel authentication solution that combines a classical single-factor solution with the exchange of a second factor using the GSM/3G/4G communication infrastructure of the user's mobile phone. By generalizing the definition in [6] for any two-factor authentication solution involving n instance factors, we have that, under the security assumptions and a set of threat-model assumptions, a two-factor authentication solution involving n instance factors is expected to meet the following security property:

(G_n) Goal G holds even if an attacker compromises up to $n-1$ instance factors.

Thus, the addition of instance factors ensures some "redundancy", meaning that if G_n holds there are no attacks even if some (but not all the) instance factors are compromised.

Considering a single *User* authentication attempt for each protocol run (which is a reasonable assumption), we want to evaluate whether the attackers of Table 5 are able to violate the G_n property. In particular, we are going to check if a protocol guarantee G_n, i.e., is a multi-factor authentication protocol, when $|IF_P(A)| < n$. To automatize the analysis, in Section 5 we provide a formalization of the attackers and the protocol in ASLan++.

4.8 Definition of Goal G_{OTP}

A main characteristic of our reference models is the use of OTPs. An OTP generator usually takes as input a set of instance factors (say *instance factors linked with OTP*, IF_P^{OTP}) and gives as output an OTP. As discussed in Section 2, an OTP "should be non-reusable and non-replicable." Indeed, if the OTP is not fresh, then the knowledge of an OTP leads to the same attacks possible when knowing the instance factors linked with it. Thus, it is crucial that the "non-reusable and non-replicable" property of the OTP is satisfied.

(G_{OTP}) Goal G holds even if an attacker compromises all the instance factors apart from at least one linked with OTP and obtains the OTP only after its use.

Thus, the use of an OTP ensures that *IdTP* accepts only one OTP for a specific operation avoiding replay attacks.

For the channel assumptions that we have defined in Section 4.4, OTP values cannot be stolen (e.g., a network attacker is not able to steal this value as a TLS channel is used). However, to assess the validity of G_{OTP}, we need to consider an attacker able of obtaining the OTP value. Thus, we consider an attacker, called *OTP Eavesdropping* (*OE*), who can indirectly[10] compromise the set of instance factors linked with the OTP. We will consider the possibility of performing this operation, called *obtain*(*OTP*), after the OTP is used to authenticate *User*.

5 Formal Specification RM_{TOTP} and RM_{CR}

In this section, we describe how the semi-formal description of RM_{TOTP} and RM_{CR} can be translated into a formal specification (in this case, specified in ASLan++), detailing the initial state, the behavior of the entities, the channels, the security assumptions, the attackers and the security goals.

5.1 Initial State

The initial state of a formal specification defines the initial knowledge of the attacker, who is indicated with the letter `i` (from the word "intruder"), and of all the honest entities that participate in the protocol run, where a protocol run is a particular run of the protocol, played by specific entities, using specific instances of the communication channels and, optionally, additional parameters that must be passed as initial knowledge to the different entities.

For what concerns the registration phase, we have modeled the data provided by the SP_{app} developer as initial knowledge of *IdTP*. In general, after the registration phase, *IdTP* creates a database storing the relation between the SP_{app} identities, either their *key_hash* values for RM_{TOTP} or *redirect_uri* for RM_{CR}, and the information (e.g., name and logo) provided by the SP developers. We have modeled the data obtained as result of the activation phase (*token_IdP* and data required for generating OTPs) as initial knowledge of *User*, *IDOTP* and *IdTP* for RM_{TOTP} and *User*, *OTPApp* and *eIDCard* for RM_{CR}. In particular, for RM_{TOTP}, the activation phase entails: a *User* knows her *PIN* value (`pinUser`), *IDOTP* knows `token_IDP` and {|`seed`|}_`pinUser`, and *IdTP* creates a DB (`usersDB`) with *User*, `token_IDP` and `seed` as entry. In RM_{CR}, the activation phase entails: *OTPApp* knows the user *PIN* value of *eIDCard* and stores the second part (`pin2`).

To specify that the attacker knows a message m, we use the ASLan++ predicate `iknows(`m`)`.

5.2 Behavior of Entities and Channels

The behavior of the honest entities is specified by the evolution of the system, which consists of a sequence of operations performed by each role. For simplicity, Fig. 5 and Fig. 6 show the evolution of the protocol using a process view, which describes the messages exchanged in Fig. 3 and Fig. 4, respectively, for each entity as a set of actions (e.g., receive or send a message and DB access), where `Actor` is the keyword used in ASLan++ to represent the entity taken into consideration. This formal representation can be translated into various role-based formal languages and provided as input to different state-of-the-art security protocol analyzers (e.g., Tamarin and ProVerif). In our analysis, given our expertise and past experience we use ASLan++ and SATMC (see [8] for more details on the language and tool).

For a detailed definition of the properties of channels between two protocol entities A and B we point the reader to [5, 38]. In a nutshell, consider a message m sent on a channel *A2B* from A to B. We write:

- `authentic_on(`*A2B*`,`A`)`, if B can rely on the fact that only A could have sent m,

[10] Indirectly means that the attacker does not actually know or possess an *IFactor* — thus he cannot generate new OTPs.

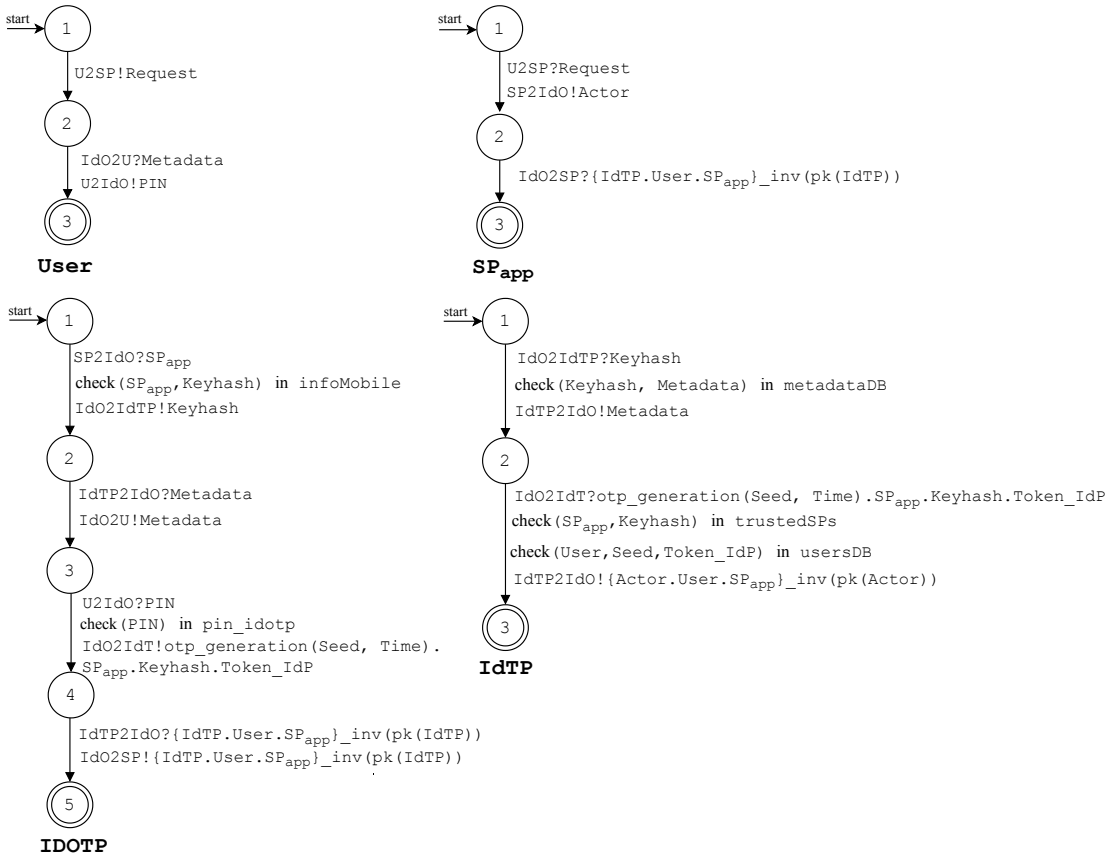
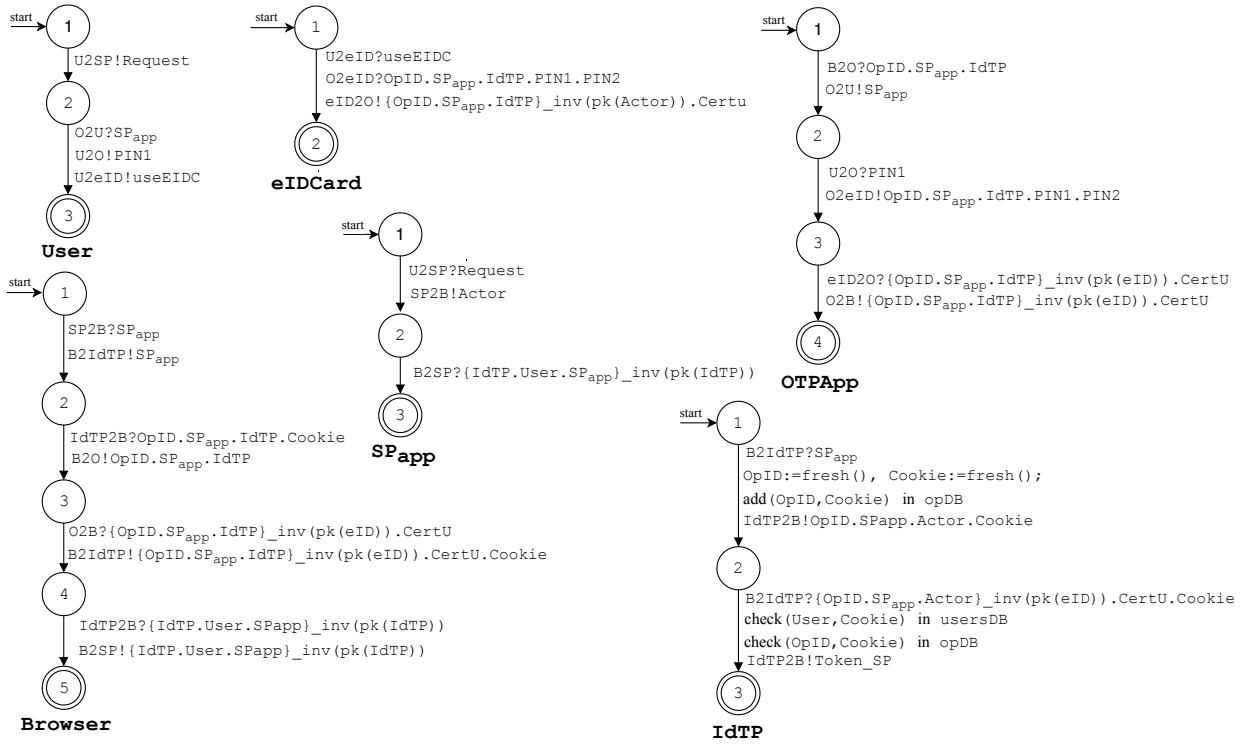


Figure 5: Protocol View for RM_{TOTP}.

- `confidential_to(`*A2B*`,`*B*`)`, if A can rely on the fact that only B can receive m,
- `weakly_authentic(`*A2B*`)`, if the channel input is exclusively accessible to a single, but yet unknown, sender, and
- `weakly_confidential(`*A2B*`)`, if the channel output is exclusively accessible to a single, but yet unknown, receiver.

A *link* property between two channels *A2B* and *B2A* (denoted `link(`*A2B*`,`*B2A*`)`) means that the entity sending messages over *A2B* is the same entity that receives messages from *B2A*. These properties can be represented graphically as follows (see Figures 3 and 4): $A \bullet\!\!\rightarrow B$, $A \circ\!\!\rightarrow B$, $A \rightarrow\!\!\bullet\, B$, $A \rightarrow\!\!\circ\, B$ mean authentic, weakly authentic, confidential and weakly confidential channel, respectively; moreover, we indicate a link property between two channels with the same arrow style.



Legend:

- U, B, O, eID stands for User, Browser, OTPApp, eIDCard respectively, and *U2SP*, B2SP, B2IdTP, SP2B, IdTP2B, O2eID, eID2O, O2B, B2O, U2O, O2U, U2eID are their unidirectional channels.
- Ch!M means that message M is sent over channel Ch.
- Ch?M means that message M is received over channel Ch.
- M1.M2 is the concatenation of messages M1 and M2.
- check(X,Y,...,Z) in DB means that (X, Y,...,Z) must be in DB, otherwise the protocol stops.
- {M}_inv(pk(A)) means that message M is encrypted with the private key of A

Figure 6: Protocol View for RM_{CR}.

5.3 Formal Specification of Assumptions

For each security assumption *sa* described in Section 4.4, Table 7 shows the corresponding formal specification $[\![sa]\!]$.

To model the TA1 and TA2$_{CR}$ assumptions, in our analysis we have not considered protocol runs with the intruder i playing the role of *IdTP* and *Browser*, respectively.

To model the communication assumptions (CA1$_{TOTP}$-CA3$_{TOTP}$ and CA1$_{CR}$-CA4$_{CR}$), we have used channel properties; as the modeling of these assumptions is far from a trivial mapping, it requires an explanation.

CA1$_{TOTP}$. It is related to the inter-app communication in the mobile. The property expected by the `StartActivityForResult` method can be modeled by a link property between the two channels used in the mobile: the app that has sent a request is the same app that will receive the result.[11]

CA2$_{TOTP}$. We have modeled the Android method, which extracts the *key_hash* value included in the package of an app, using an authentic channel (used by SP_{app} to send its identity to *IDOTP*) and a DB

[11] `StartActivityForResult` guarantees also that SP_{app} sends a request to right *IDOTP* app. However, as we are considering the minimal set of assumptions that are necessary to prevent a violation of the authentication property, we have modelled CA1$_{TOTP}$ only with the link property. Indeed, even if an SP_{app} invokes a malicious app (instead of the right *IDOTP*), there is no violation of the authentication property, as the malicious app does not have the seed value needed to generate valid OTPs.

Table 7: Mapping between Security Assumptions (Asm(s) for short) and Formal Specification.

Security Asms (sa)	**Formal Specification** ($[\![sa]\!]$)
TA1	We do not consider protocol runs with `i` playing the role of *IdTP*
TA2$_{CR}$	We do not consider protocol runs with `i` playing the role of *Browser*
CA1$_{TOTP}$	`link(`*SP2IdO*`,`*IdO2SP*`);`
CA2$_{TOTP}$	`authentic_on(`*SP2IdO*`,`SP_{app}`);`
CA3$_{TOTP}$	`confidential_to(`*IdO2IdTP*`,`*IdTP*`);` `weakly_authentic(`*IdO2IdTP*`);` `weakly_confidential(`*IdTP2IdO*`);` `authentic_on(`*IdTP2IdO*`,`*IdTP*`);` `link(`*IdO2IdTP*`,`*IdTP2IdO*`);`
CA1$_{CR}$	`confidential_to(`*B2SP*`,`SP_{app}`);`
CA2$_{CR}$	`confidential_to(`*B2IdTP*`,`*IdTP*`);` `weakly_authentic(`*B2IdTP*`);` `weakly_confidential(`*IdTP2B*`);` `authentic_on(`*IdTP2B*`,`*IdTP*`);` `link(`*B2IdTP*`,`*IdTP2B*`);`
CA3$_{CR}$	`confidential_to(`*O2B*`,`*Browser*`);`
CA4$_{CR}$	`confidential_to(`*O2eID*`,`*eIDCard*`);` `weakly_authentic(`*O2eID*`);` `weakly_confidential(`*eID2O*`);` `authentic_on(`*eID2O*`,`*eIDCard*`);` `link(`*O2eID*`,`*eID2O*`);`
ActivA	Data obtained during the activation phase are `nonpublic` values and the communication channels used are secure

containing the relations between the SP_{app} identities and their *key_hash*, used by *IDOTP* to read the correct *key_hash* value. This is due to the fact that this method — executed by the Android OS — guarantees the authenticity of its output.

CA3$_{TOTP}$. It is modeled with five channel properties (see Table 7) that all together model a TLS unilateral channel (see [8] for more details).

CA1$_{CR}$. The property expected by the redirection scheme specified in CA1$_{CR}$ (Android App Links) can be modeled with a confidential channel between *Browser* and SP_{app}.

CA2$_{CR}$. We have modeled CA2$_{CR}$ with five channel properties (see Table 7) that all together model a TLS unilateral channel.

CA3$_{CR}$. We have modeled CA3$_{CR}$, which represents the launching of a browser after the OTP generation, as a confidential channel between *OTPApp* and *Browser*.

CA4$_{CR}$. The idea underlying CA4$_{CR}$ is that *eIDCard* and *OTPApp* establish a secure channel before the challenge exchange of Step A9 of Fig. 4. This assumption can be modeled with five channel properties (see Table 7) that all together model this secure channel.

Finally, to model ActivA we have set a `nonpublic` label to the instance factors obtained from the activation phase.

5.4 Formal Specification of Attackers

In our analysis, we have considered the behavior of a Dolev-Yao (DY) intruder [14], who cannot break cryptography but can overhear and modify messages using his initial knowledge and the knowledge obtained from the traffic. This behavior is usually built-in in security protocol analysis tools (like SATMC). In addition to the classic DY intruder, as described in Section 4.6, we have considered a set of attackers with different capabilities able to compromise the authentication instance factors. In the following, we describe in detail how we have formalized in ASLan++ $own(IFactor_o)$, $interact(E)$, $interact_{ME}(E)$, $know(IFactor_k)$

and *overhear*(E), where an $IFactor_o$ corresponds to a `nonpublic` value stored by E and an $IFactor_k$ is a `nonpublic` value entered by *User* into E. Table 8 shows this formalization, where on the $[\![a]\!]$ column there are the properties that we have to add to the model when considering a specific attacker, on the $[\![\bar{a}]\!]$ column the ones to add when an attacker is not present. For the sake of brevity we remove the universal quantifiers related to the operations, which are reported in Table 6.

We have formally modeled the compromising of an *IFactor* as follows.

- *know*($IFactor_k$): we give the $IFactor_k$ to the attacker as initial knowledge, i.e., we add

  `iknows(`$IFactor_k$`);`

- *own*($IFactor_o$): we give the $IFactor_o$ to the attacker as initial knowledge, i.e., we add

  `iknows(`$IFactor_o$`);`

We have formally modeled the compromising of an entity as follows.

- *overhear*(E): to model whether (or not) the $IFactor_k$ is compromised during the protocol run we consider (or not) the following property of the channel $User2E$:

  `confidential_to(`$User2E$`,`E`);`

  In particular, we add the confidential channel property in case we assume the protection against this attacker.

- *interact*(E): to model whether (or not) the $IFactor_o$ is compromised during the protocol run we consider (or not) the following property of the channel $X2E$ with X an honest entity:

  `authentic_on(`$X2E$`,`X`);`

  In particular, we add the authentic channel property in case we assume the protection against this attacker.

- $interact_{ME}$(E): to model whether (or not) the $IFactor_o$ is compromised during the protocol run in a mutual exclusive way, we consider (or not) the following property of the channel $X2E$ with X an honest entity:

  `weakly_authentic(`$X2E$`);`

  In particular, we add the weakly authentic channel property when we want to consider this attacker in the analysis. Usually, when we are modeling a stolen entity, X is played by the *User*. In our reference models, we have two physical entities that can be stolen: the smartphone and the electronic identity card. For the latter, E is played by the card, whereas for the smartphone, as we do not have an entity playing the device as a whole, we need to introduce a fact:

  `UserOwnSmartphone`;

  We add this fact as a pre-condition for the execution of the *User*'s actions. This fact is present (meaning it is true) if the smartphone is not stolen (and so the *User* will behave honestly), and must be deleted (meaning it is false) otherwise (and so the *User* will not interact with any entities on the smartphone).

Finally, we have formally modeled the compromising of an OTP value as follows:

- *obtain*(OTP): we give to the attacker the knowledge of the OTP, i.e., we add

  `iknows(OTP);`

  at State 3 of *IdTP* when it is used after *IdTP* accepts it (see Fig. 5 and 6).

Table 8: Formal Specification of the Attacker.

Attacker Operation a		**Formal Specification** $[\![\bar{a}]\!]$ (w/o attacker)	$[\![a]\!]$ (with attacker)
$DT_{\square}$	$interact_{ME}(E)$	`authentic_on(`$User2E$`,`$User``);`	`weakly_authentic(`$User2E$`);`
		`UserOwnSmartphone;`	-
$DT_{\blacksquare}$	$interact_{ME}(\blacksquare)$	`authentic_on(`$User2\blacksquare$`,`$User``);`	`weakly_authentic(`$User2\blacksquare$`);`
$SE \vee SS$	$know(IFactor_k)$	-	`iknows(`$IFactor_k$`);`
AD	$own(IFactor_o)$	-	`iknows(`$IFactor_o$`);`
ES	$overhear(E)$	`confidential_to(`$User2E$`,`$E`);`	-
MA	$interact(\blacksquare)$, with X honest entity	`authentic_on(`$X2\blacksquare$`,`$X`);`	-
	$overhear(E)$, with X honest entity	`confidential_to(`$User2E$`,`$E`);`	-
	$interact(E)$, with X honest entity	`authentic_on(`$X2E$`,`$X`);`	-
	$own(IFactor_o)$	-	`iknows(`$IFactor_o$`);`

5.5 Formal Specification of Security Goals

As described in Section 4.7 and Section 4.8, we have defined G_n and G_{OTP} in terms of a traditional authentication goal, the security assumptions and the attackers. This means that, in the formal specification, we consider the traditional authentication goal G and we check whether it holds under the security assumptions and the different set of attackers. The property must hold if the set of attackers is not able to compromise all the instance factors.

G requires that a message is transmitted in an authenticated and fresh manner, thus allowing SP_{app} to authenticate *User* and offering replay-protection at the same time. For the definition of authentication we refer to the definition of *non-injective agreement* in [35]: we say that SP_{app} *authenticates User on message* M if whenever the entity SP_{app} completes a run of the protocol apparently with the entity *User*, then *User* has previously been running the protocol apparently with SP_{app}, and the two entities agree on M. In addition, to ensure replay protection, we require freshness. In ASLan++, this is formalized by specifying the goal:

(G) `SP_authn_U_on_M:(_)` *User* `*->>` SP_{app};

where `*->>` indicates authenticity, directedness (i.e., the only (honest) receiver of a message is the intended one [8]) and freshness. In addition, following the definition in [35], associated goal labels are used to specify which values of M the goal is referring to, namely, the `Request` value in State 1 of the *User* process (in Fig. 5 and Fig. 6) and the corresponding value in the last state of the SP_{app} process (State 3 in Fig. 5 and Fig. 6). For more details about the goal channel semantics, please refer to [8].

6 Security Analysis of RM_{TOTP} and RM_{CR}

Our focus is determining whether the concurrent execution of protocol runs enjoys the expected security goals in spite of the presence of one attacker or a set of attackers.

We consider model checking problems of the form:

$$I, M, [\![SA_M]\!], Asms_A \models G \tag{1}$$

where

- I is the initial knowledge of the honest entities and the attacker (cf. Section 5.1).
- M is a state transition system modeling the behavior of the entities. In our analysis it can be either RM_{TOTP} or RM_{CR}, sketched in Fig. 5 and Fig. 6, respectively.
- $[\![SA_M]\!]$: given the security assumptions $\{sa_1, \ldots, sa_m\}$ of M reported in Table 7, we define $[\![SA_M]\!]$= $\{[\![sa_1]\!], \ldots, [\![sa_m]\!]\}$ to be the set of security assumptions that formalize them.
- $Asms_A$: while the behavior of the DY intruder is built-in in the tool and we thus do not explicitly model it, we consider sets of attacker behaviors, reported in Table 8. Given a set of attacker behaviors $A = \{a_1, \ldots, a_j\} \subseteq TM$, we define $Asms_A$=$[\![DY|^A]\!] \cup [\![DY|_{\bar{A}}]\!] = \{[\![a_1]\!], \ldots, [\![a_j]\!], [\![\bar{a}_{j+1}]\!], \ldots, [\![\bar{a}_7]\!]\}$, where $\bar{A}$=$TM \setminus A$= $\{a_{j+1}, \ldots, a_7\}$. This means that we empower a DY intruder with the properties formalized in $[\![a_1]\!], \ldots, [\![a_j]\!]$, while we limit his capabilities adding the properties $[\![\bar{a}_{j+1}]\!], \ldots, [\![\bar{a}_7]\!]$.
- G: the goal is always G as defined in Section 5.5. Indeed, to evaluate G_n and G_{OTP} we change the assumptions.

For both reference models we performed the following analyses:

Analysis on Security Assumptions: in this analysis we do not consider any attacker behavior and we remove from the formal specification a security assumption at a time. Namely, with respect to (1) we consider *(i)* $A = \emptyset$, and *(ii)* instead of $[\![SA_M]\!]$, each $[\![SA'_M]\!]$ such that $SA'_M \subset SA_M$ and $|SA'_M| = |SA_M| - 1$. The goal of this analysis is to prove that all the security assumptions reported in Table 7 are necessary to prevent a violation of the security goal G.

Analysis on G_n: given a solution involving n instance factors, to evaluate the goal G_n we consider all the sets of attackers A among the ones listed in Table 6 such that $|IF_P(A)| < n$.

Analysis on G_{OTP}: to evaluate the goal G_{OTP}, we consider sets of attackers A such that all the instance factors apart from at least one linked with OTP are compromised and the OTP is obtained only after its use. Namely, with respect to (1), let us call IF_P^{OTP} the set of instance factors in P linked with OTP. We consider each A such that $IF_P(A) \supseteq (IF_P \setminus IF_P^{OTP})$ and $IF_P(A) \not\supseteq IF_P^{OTP}$.

Given our expertise and a positive past experience, we have selected SATMC among the available state-of-the-art tools for the formal analysis of security protocols. The complete set of specifications and figures (in this paper, for the sake of clarity, we represent only the significant steps of the attack traces reported by SATMC) can be found at the companion website.[12]

SATMC, like all other state-of-the-art security protocol analysis tools, has some inherent limitations, which, however, did not hinder or weaken our analyses.[13] For instance, SATMC requires the user to fix a number of parallel protocol runs. In all our analyses, we have considered several scenarios including (at most) three parallel protocol runs in which the attacker either does not play any role or plays the role of SP_{app}. Moreover, SATMC carries out an iterative deepening strategy on k. Initially k is set to 0, and then it is incremented until an attack is found (if any) or k_{max} is reached. If this is the case, then no attack traces of length up to k_{max} exist. The trace includes the actions performed by the attacker and the honest participants. The length of the trace is counted by taking into account the parallel execution of non-conflicting actions (actions executed in parallel are considered as a single step). Notice that most of the actions of the attacker are executed in parallel (and counted as a single step) with the ones of honest participants. We set k_{max} to 1.5 times the length of the longest trace of the protocol when only honest entities participate. This heuristic choice arises from past experience in security protocol analysis [4, 7] and seems to offer a good trade-off between efficiency (i.e. keeping the time and memory consumption of the tool to a reasonable level) and confidence that a large enough portion of the search space has been explored to detect possible attacks (i.e. longer execution traces are unlikely to unveil attacks). In our analysis, the length of the longest trace of the protocol when only honest entities participate is given by the total number of states for the involved entities of Fig. 5 and Fig. 6 and corresponds to 14 and 20, respectively.

[12] `https://stfbk.github.io/complementary/TOPS2020`

[13] Still, as future work, it could be interesting to perform similar analyses using other tools or extensions of SATMC.

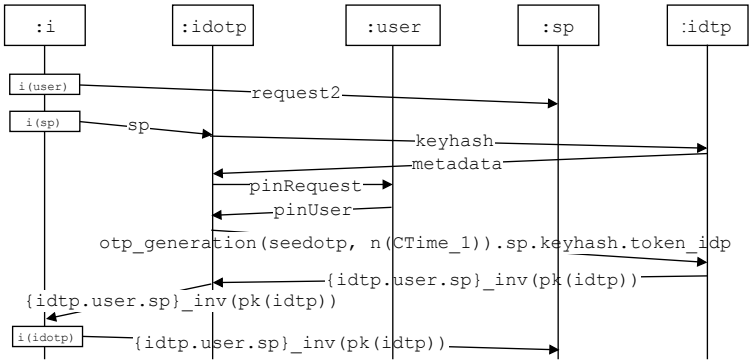


Figure 7: Attack trace without the security assumption CA2$_{TOTP}$.

Table 9: Results for RM_{TOTP}.

Id	Set of Attackers A	PIN	{seed}_PIN	token_IdP
1	$DT_{\square}$ (*)	🔒	🔓	🔓
2	$SE \vee$ SS	🔓	🔒	🔒
3	AD (*)	🔒	🔓	🔓
4	ES	🔓	🔒	🔒
5	$DT_{\square} \wedge AD$ (*)	🔒	🔓	🔓
6	$(SE \vee$ SS$) \wedge ES$	🔓	🔒	🔒

6.1 Results for RM_{TOTP}

We give below the main details of our analyses for M=RM_{TOTP}:

Analysis on Security Assumptions: given that the security assumptions for RM_{TOTP} are TA1, CA1$_{TOTP}$, CA2$_{TOTP}$, CA3$_{TOTP}$ and ActivA, we have performed 5 executions of SATMC removing one security assumption at a time. SATMC always finds an attack. To provide an example, Fig. 7 shows the attack trace obtained by removing CA2$_{TOTP}$. In this attack, `i` initiates a protocol run with `idotp` pretending to be `sp` (indicated as `i(sp)`). This is permitted as the channel used is not authentic; thus, `i` can pretend to be another app. Then, given the link property of the channels between `i` and `idotp`, `i` is able to obtain a token_SP (namely, `{idtp.user.sp}_inv(pk(idtp))`) and impersonate the *User* to the actual `sp`.

Analysis on G_n: Table 9 reports (concisely, by using logic operators) all the possible sets of attackers such that up to $n-1$ instance factors are explicitly compromised. In these cases, SATMC did not find any attack, thus we can conclude that RM_{TOTP} satisfies G_n under our assumptions. Note that MA is not present in Table 9 as MA alone is able to compromise all the instance factors. This is possible as only one device is used. Thus, if the device is compromised, in this case by a malicious app able to obtain root privileged permissions, then the attacker is able to obtain a valid identity assertion and use it to impersonate the *User* to an honest SP_{app}. However, by instructing the *User* to download apps from the official store and to update the version of the operating system, the likelihood to have a malicious app with root privilege permissions is low. Indeed, as stated in the *Android Security & Privacy 2018 Year In Review* [2], newer versions are "more resilient to privilege escalation attacks that previously allowed potential harmful applications to gain persistence on devices and protect themselves against removal attempts." For this reason, we are confident in the practical security of the solution.

Analysis on G_{OTP}: given that $IF^{OTP}_{RM_{TOTP}}$= $\{PIN, \{|seed|\}\_PIN\}$, from Table 9, we can observe that the sets of attackers (indicated with (*)) such that all the instance factors apart from at least one linked with

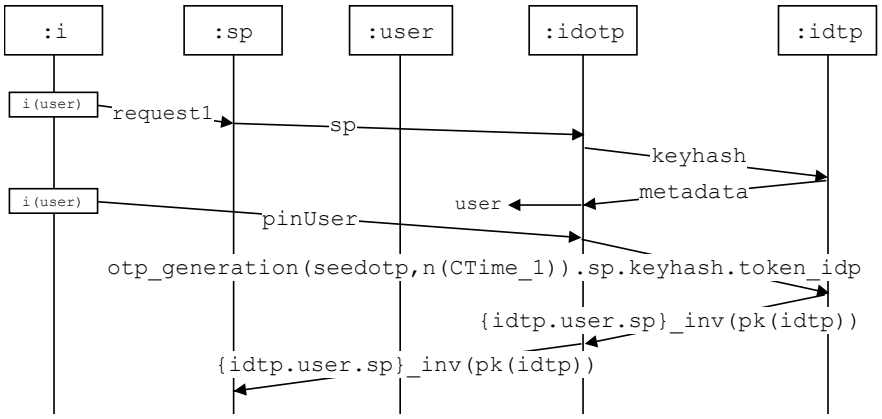


Figure 8: Attack trace obtained by considering $DT_{\square} \wedge SS$.

OTP (in this case, the PIN value) are compromised are $DT_{\square}$, AD and $DT_{\square} \wedge AD$. For these sets of attackers, SATMC does not find any attack when the OTP value is obtained after its use, thus we can conclude that RM_{TOTP} satisfies G_{OTP} under our assumptions.

As expected, in the analyses on G_n and G_{OTP}, SATMC did not find any attack. To prove that this result in not given by the setting of restricted assumptions but by the capabilities of the attacker, we discuss two examples where a more powerful attacker is able to compromise all the instance factors or to use the stolen OTP before the user. In these cases, SATMC returns the expected attack traces. More specifically:

- Fig. 8 shows the attack trace obtained by considering the set $DT_{\square} \wedge SS$, that is, for example, an attacker that watches the *PIN* entered by *User* (SS) and then steals the smartphone ($DT_{\square}$). In the attack, `i` initiates a protocol run with `sp` pretending to be `user` (indicated as `i(user)`). By entering the PIN code (`pinUser`) when requested by `idotp`, `i` is able to impersonate the *User* and obtain the requested resource (`resources1`).
- Fig. 9 shows the attack trace obtained by considering the set $AD \wedge OE$, that is, an attacker that extracts the *token_IdP* value from `idotp` and is able to intercept the OTP value before its use. In this attack, `i` initiates a protocol run with `sp` pretending to be `user` (indicated as `i(user)`). Then, by sending the OTP value together with the stolen token to `idtp`, `i` is able to obtain a token_SP {`idtp.user.sp`}_`inv(pk(idtp))` and use it to finalize the authentication process to the honest `sp`.

6.2 Results for RM_{CR}

We give below the main details of our analyses for $M{=}RM_{CR}$:

Analysis on Security Assumptions: given that the assumptions for RM_{CR} are TA1, TA2_{CR}, CA1_{CR}, CA2_{CR}, CA3_{CR}, CA3_{CR} and ActivA, we have performed 7 executions of SATMC removing one security assumption at a time. SATMC always finds an attack. Fig. 10 shows the attack trace obtained by removing CA1_{CR}. In this attack, `i` is able to obtain a token_SP (namely, {`idtp.user.sp`}_`inv(pk(idtp))`) and reuse it in his smartphone. As a consequence, `i` is authenticated by `sp` as the victim (`user`). This attack takes advantage of a wrong implementation of the redirection from *Browser* to SP_{app}.

Analysis on G_n: Table 10 shows all possible sets of attackers such that up to $n-1$ instance factors are compromised. In these cases, SATMC was not able to find attacks, thus we can conclude that RM_{CR} satisfies G_n under our assumptions. As before MA is not present in Table 10 as MA alone is able to compromise all the instance factors.

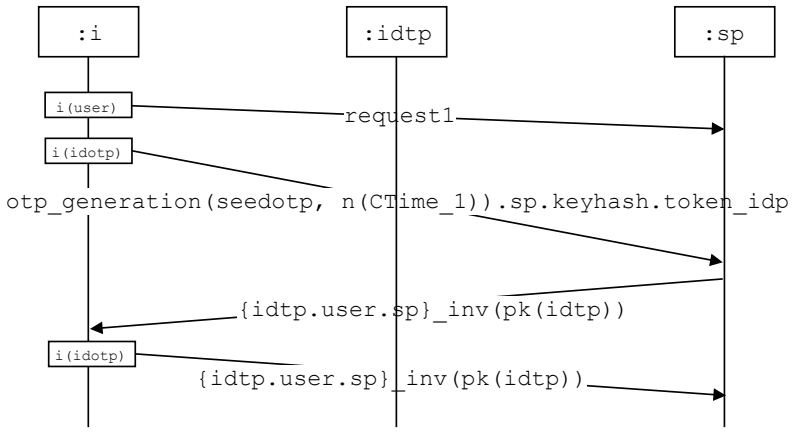


Figure 9: Attack trace obtained by considering $AD \wedge OE$ (before its use).

Table 10: Results for RM_{CR}.

Id	Set of Attackers	PIN1	PIN2	eIDCard
1	$DT_{\square}$	🔒	🔓	🔒
2	$DT_{\blacksquare}$	🔒	🔒	🔓
3	$SE \vee \text{SS}$	🔓	🔒	🔒
4	AD	🔒	🔓	🔒
5	ES	🔓	🔒	🔒
6	$DT_{\square} \wedge DT_{\blacksquare}$	🔒	🔓	🔓
7	$DT_{\square} \wedge (SE \vee \text{SS})$	🔓	🔓	🔒
8	$DT_{\square} \wedge AD$	🔒	🔓	🔒
9	$DT_{\square} \wedge ES$	🔓	🔓	🔒
10	$DT_{\blacksquare} \wedge (SE \vee \text{SS})$	🔓	🔒	🔓
11	$DT_{\blacksquare} \wedge AD$	🔒	🔓	🔓
12	$DT_{\blacksquare} \wedge ES$	🔓	🔒	🔓
13	$(SE \vee \text{SS}) \wedge AD$	🔓	🔓	🔒
14	$(SE \vee \text{SS}) \wedge ES$	🔓	🔒	🔒
15	$AD \wedge ES$	🔓	🔓	🔒
16	$DT_{\square} \wedge DT_{\blacksquare} \wedge AD$	🔒	🔓	🔓
17	$DT_{\square} \wedge (SE \vee \text{SS}) \wedge AD$	🔓	🔓	🔒
18	$DT_{\square} \wedge (SE \vee \text{SS}) \wedge ES$	🔓	🔓	🔒
19	$DT_{\square} \wedge AD \wedge ES$	🔓	🔓	🔒
20	$DT_{\blacksquare} \wedge (SE \vee \text{SS}) \wedge ES$	🔓	🔒	🔓
21	$(SE \vee \text{SS}) \wedge AD \wedge ES$	🔓	🔓	🔒
22	$DT_{\square} \wedge (SE \vee \text{SS}) \wedge AD \wedge ES$	🔓	🔓	🔒

Analysis on G_{OTP}: we can observe that all the instance factors of RM_{CR} are linked to the OTP generation (namely, $IF^{OTP}_{RM_{CR}} = \{PIN1, PIN2, eIDCard\}$). Thus, for all sets of attackers in Table 10 we have checked G_{OTP}. As a result, we have that for all these sets SATMC does not find attacks when the OTP value is obtained after its use, thus we can conclude that RM_{CR} satisfies G_{OTP} under our assumptions.

As for RM_{TOTP}, to prove that this result in not given by the setting of restricted assumptions but by the capabilities of the attacker, we discuss an attack trace found by SAMTC in case of a powerful attacker.

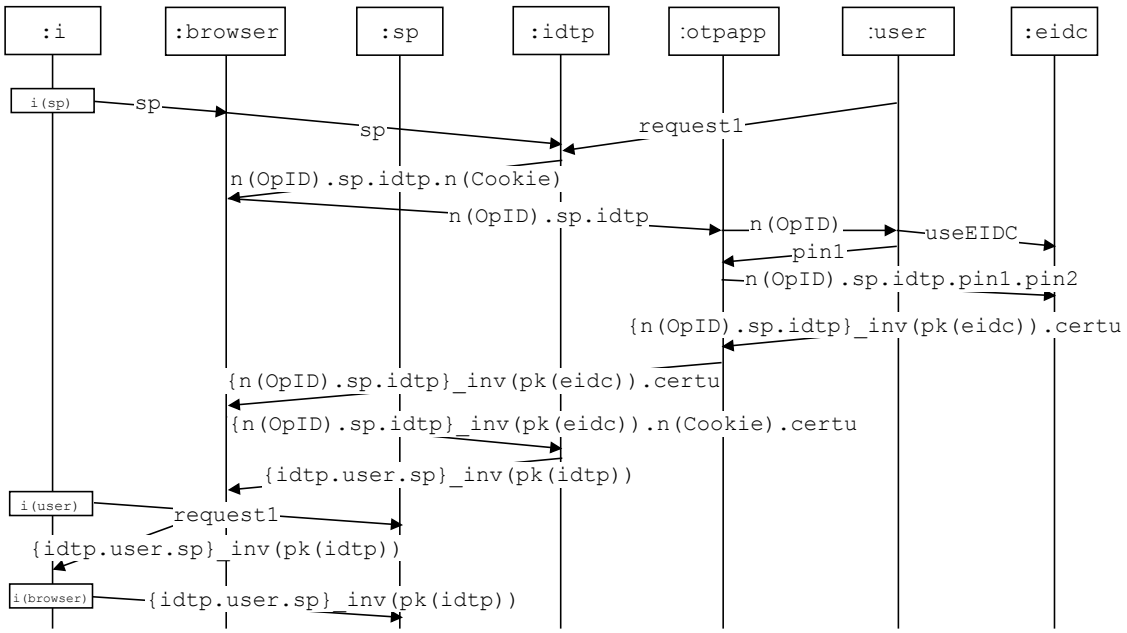


Figure 10: Attack trace without the security assumption CA1_{CR}.

Fig. 11 shows the attack trace obtained by considering the set $DT_{\square} \wedge DT_{\blacksquare} \wedge \text{SS}$, that is, for example, an attacker that watches the *PIN1* entered by *User* and then steals the smartphone and the user's *eIDCard*. In the attack, `i` sends a request (`request1`) to `sp` pretending to be `user` (indicated as `i(user)`). By entering the half PIN code (`pin1`) and using the *eIDCard* when requested by `otpapp`, `i` is able to impersonate the `user`.

6.3 Summary

The security analyses that we have performed (summarized in Table 11) allowed us to confirm that the security assumptions of Section 4.4 are necessary to prevent trivial attacks, and the two reference models that we have proposed in Section 4.3 satisfy the expected authentication properties. In detail, for both reference models, we have that:

Analysis on Security Assumptions: by removing from the formal specification only one of the security assumptions at a time (indicated, for short, as "all−1" in Table 11) we have a violation of G. Given that an attack is found whenever a single assumption is removed, we have that the $[\![SA_M]\!]$ set is minimal with respect to the analyzed scenario.

Analysis on G_n: by considering sets of attackers A among the ones listed in Table 6, SATMC does not find any attack on the solution if up to $n-1$ instance factors are explicitly compromised.

Analysis on G_{OTP}: SATMC does not find any attack on the solution if all the instance factors apart from at least one linked with OTP are compromised and the OTP is obtained only after its use.

These results are useful per se but also as a guideline and basis for future similar specifications and analyses. For instance, our reference models could be adopted by designers of solutions that fall within the requirements of our scenarios and guarantee the same security assumptions; this allows for the re-use of the existing infrastructures (e.g., SAML-based IdPs) in the mobile context. In general, our modeling and analysis methodology can be used directly when considering other MFA protocols or, when necessary, it can be extended by adding other attackers or the support of the multi-factor authorization scenario.

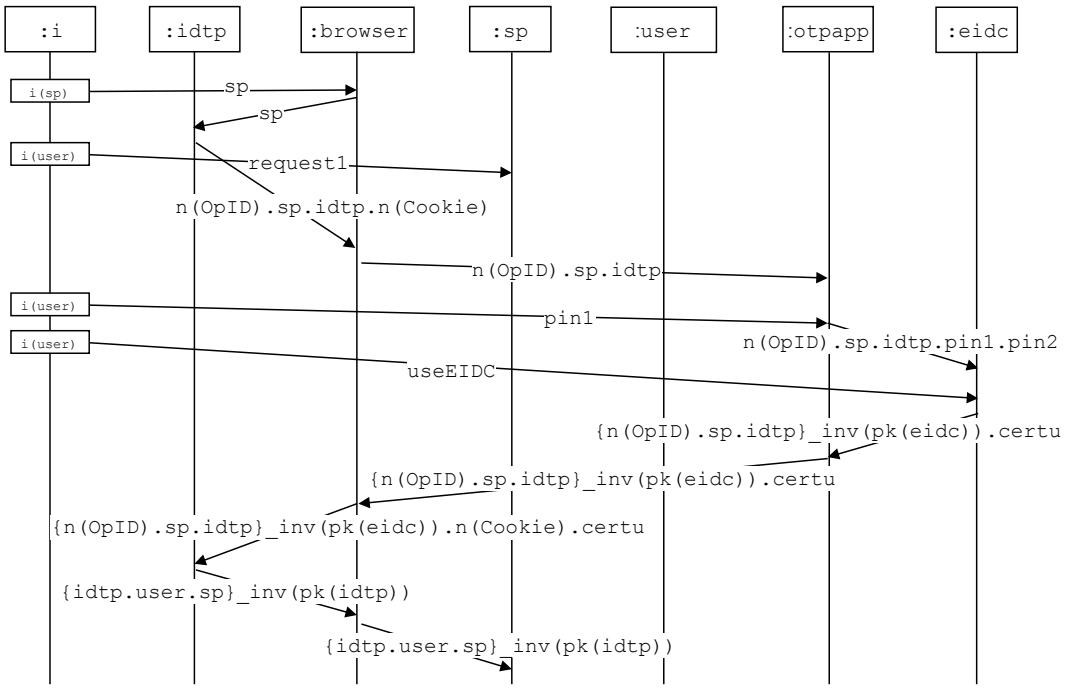


Figure 11: Attack trace obtained by considering $DT_{\square} \wedge DT_{\blacksquare} \wedge \text{SS}$.

Table 11: Analyses performed.

Analysis on	$[\![SA_M]\!]$	**Attackers** A	**Atk on** RM_{TOTP}	**Atk on** RM_{CR}
Security Assumptions	all -1	$\emptyset$	Yes	Yes
G_n	all	A such that $\lvert IF_P(A)\rvert < n$	No	No
G_{OTP}	all	A s.t. $IF_P(A) \supseteq (IF_P \setminus IF_P^{OTP})$ and $IF_P(A) \not\supseteq IF_P^{OTP}$, and `iknows(OTP)` after its use	No	No

7 Related Work

SSO Protocols for Native Apps. OAuth 2.0 [29] and OpenID Connect [41] have been designed for light-RESTful API services, and are considered the de-facto standards for managing authentication and authorization. These protocols are well-accepted in the web scenario, but they provide only partial support for mobile apps (there is a frequent use of the expression "out of scope" in [29, 41]). This could lead to the implementation of insecure solutions. An in-depth analysis of OAuth in the mobile environment — underlining possible security problems and vulnerabilities — is available in [13, 48]. Given the lack of specifications, the OAuth Working Group has released in 2017 a best practice with the title "OAuth 2.0 for Native Apps" [40]. As described in Section 4, our reference model RM_{CR} generalizes the one proposed in [40].

Formal Security Analysis: Web and Mobile. Although these techniques are useful for the analysis of a specific implementation (as they are able to discover serious security flaws), it is important to perform a comprehensive security analysis of the standard itself. In the context of web apps, Fett et al. [21] performed a formal analysis of the OAuth protocol using an expressive web model (defined in [20]) that describes

the interaction between browsers and servers in a real-world set-up. This formal analysis revealed two unknown attacks on OAuth that violate the authorization and authentication properties. A similar analysis is performed for OpenID Connect in [22]. Two other examples of formalizations of OAuth are [9], where the different OAuth flows are modeled in the Applied Pi calculus and verified using ProVerif extended with WebSpi (a library that models web users, apps and intruders), and [43], where OAuth is modeled in Alloy.

In our analysis (cf. Section 5) we used ASLan++ and SATMC. In the past, SATMC has revealed severe security flaws in the SAML 2.0 protocol [39] and in the variant implemented by Google [5]; by exploiting these flaws a dishonest service provider could impersonate a user at another service provider. Moreover, Yan et al. [55] used ASLan++ and SATMC to analyze four security properties of OAuth: confidentiality, authentication, authorization, and consistency.

The aforementioned formal analyses, however, focus on the web app scenario, whereas in this paper we deal with native apps. In [57], Ye et al. used ProVerif to analyze the security of a SSO implementation for Android. They applied their approach to the implementation of the Facebook Login and identified a vulnerability that exploits super-user permissions.

Formal Security Analysis for MFA The literature related to formal analysis of MFA solutions is very recent. Jacomme et al. [33] propose a threat model for MFA protocols that combines a classic Dolev-Yao attacker with different attacker levels. The different levels are modeled as read or write access to difference input and/or output interfaces composing the system (e.g., a key logger could be modeled as a malware with read-only access to the USB input interface). This threat model is then formalized using the Applied pi calculus and the ProVerif tool. There are some common choices between [33] and our analysis, such as the modeling of an attacker capability as a property of a channel: in [33] by giving read and/or write access to a private channel, in our case with the definition of security properties (see Section 5.3). The main difference is the relation of the security analysis with authenticator factors, while in [33] they assume only the password compromised, in our analysis the instance factors have a central role. As proposed in [6], we have defined a MFA goal based on them. In [6], the authors analyzed a two-factor and two-channel authentication solution that combines a classic single-factor solution with the exchange of a second factor using the GSM/3G/4G communication infrastructure of the user's mobile phone. In this work, we have generalized the definition in [6] for any OTP generation approach and considering a solution involving n instance factors.

Among the MFA solutions on the market, *YubiKey NEO* [58] is one of the most attractive. It is a token device that supports OTPs and the FIDO Alliance Universal 2nd Factor (U2F) protocol, and, by integrating a Near Field Communication (NFC) technology, it can be used to provide a second-factor also in the mobile context. A formal analysis of FIDO U2F was carried out by Pereira et al. in [44]. Their analysis showed that ignoring an optional verification step of the standard could lead to an user implementation attack. RM_{CR} can be easily modified to support the FIDO protocol.

8 Conclusions

We have presented the design of two MFA reference models for native apps based on the requirements of two real-world use-case scenarios (TreC mHealth and CIE 3.0 eID schemes) that include an OTP exchange and provide a SSO experience. In addition to the protocol flow description, we have detailed the security assumptions and defined two security goals: G_n related to a multi-factor authentication solution and G_{OTP} that identifies the properties of an OTP. To perform a security analysis of our reference models we have formally modeled the flow, assumptions and goals using a formal language (ASLan++) and checked the identified security goals using a model-checker (SATMC) in the presence of different sets of attackers.

The solution we have presented, as well as the formal specification and analysis that we have given, can be generalized quite straightforwardly to other use cases, which we are currently doing. As future work, we also plan to extend the analysis to other authentication factors, such as biometric traits. In addition, an interesting future direction could be to establish a collaboration with researchers focused on vulnerability detection of SSO and MFA protocols (e.g., [51, 54, 50, 56] for OAuth and OIDC) in order to provide a

support for the discovery process (similarly to what has been proposed by the SPaCIoS project [52] for the Internet of Services).

Acknowledgement We thank colleagues at IPZS for their collaboration on the development of the authentication solution based on the CIE 3.0 carried out in the context of the joint laboratory DigiMat-Lab between FBK and IPZS. We also thank the people of the Health & Wellbeing high-impact initiative of FBK for their collaboration on the TreC project of the Azienda Provinciale per i Servizi Sanitari.

References

[1] Android. Handling Android App Links. `https://developer.android.com/training/app-links/index.html`, 2017.

[2] Android. Android Security & Privacy 2018 Year In Review, 2019.

[3] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, and L. Viganò. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In *Proceedings of the 18th TACAS*, pages 267–282. Springer, 2012.

[4] A. Armando, R. Carbone, and L. Compagna. SATMC: a SAT-based model checker for security protocols, business processes, and security APIs. *STTT*, 18(2):187–204, 2016.

[5] A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-On: Breaking the SAML-based Single Sign-On for Google Apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering (FMSE)*, pages 1–10, 2008.

[6] A. Armando, R. Carbone, and L. L. Zanetti. Formal Modeling and Automatic Security Analysis of Two-Factor and Two-Channel Authentication Protocols. In *Proceedings of 7th International Conference on Network and System Security (NSS)*, pages 728–734, 2013.

[7] A. Armando and L. Compagna. Sat-based model-checking for security protocols analysis. *Int. J. Inf. Sec.*, 7(1):3–32, 2008.

[8] AVANTSSAR Project. Deliverable D2.3 (update) ASLan++ specification and tutorial. `http://www.avantssar.eu/pdf/deliverables/avantssar-d2-3_update.pdf`, 2008. Also available as `https://stfbk.github.io/complementary/TOPS2020`.

[9] C. Bansal, K. Bhargavan, and S. Maffeis. Discovering Concrete Attacks on Website Authorization by Formal Analysis. In *Proceedings of 25th IEEE Computer Security Foundations Symposium (CSF'12)*, pages 247–262, 2012.

[10] D. A. Basin, C. Cremers, and C. A. Meadows. Model checking security protocols. In *Handbook of Model Checking.*, pages 727–762. 2018.

[11] BBA. An app-etite for banking. `https://www.bba.org.uk/wp-content/uploads/2017/06/WWBN-IV.pdf`, 2017.

[12] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre. ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial . `https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf`, 2018.

[13] E. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague. OAuth Demystified for Mobile Application Developers. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2014.

[14] D. Dolev and A. Yao. On the Security of Public-Key Protocols. In *IEEE Transactions on Information Theory.*, page 2(29), 1983.

[15] European Banking Authority. Final guidelines on the security of Internet payments, 2014.

[16] European Commission. Regulation EU 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (GDPR), 2016.

[17] European Parliament. Regulation 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. `http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32014R0910&from=EN`, 2014.

[18] Facebook. Getting started with the Facebook SDK for Android. `https://developers.facebook.com/docs/android/getting-started/facebook-sdk-for-android/`, 2015.

[19] S. Fahl, M. Harbach, M. Oltrogge, T. Muders, and M. Smith. Hey, You, Get Off of My Clipboard — On How Usability Trumps Security in Android Password Managers. In *Financial Cryptography and Data Security*, pages 144–161, 2013.

[20] D. Fett, R. Küsters, and G. Schmitz. An Expressive Model for the Web Infrastructure: Definition and Application to the BrowserID SSO System. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P)*, pages 673–688. IEEE Computer Society, 2014.

[21] D. Fett, R. Küsters, and G. Schmitz. A Comprehensive Formal Security Analysis of OAuth 2.0. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1204–1215. ACM, 2016.

[22] D. Fett, R. Küsters, and G. Schmitz. The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines. In *Proceedings of the 30th Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, 2017.

[23] Google. Google Authenticator. `https://support.google.com/accounts/answer/1066447?hl=en`, 2019.

[24] P. A. Grassi, F. J. L., N. E. M., R. A. Perlner, A. R. Regenscheid, W. E. Burr, and J. P. Richer. *Digital Identity Guidelines*. National Institute of Standards and Technology, 6 2017.

[25] D. He, M. Naveed, C. A. Gunter, and K. Nahrstedt. Security Concerns in Android mHealth App. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4419898/`, 2014.

[26] IETF. HOTP: An HMAC-Based One-Time Password Algorithm. `https://tools.ietf.org/html/rfc4226`, 2005.

[27] IETF. OCRA: OATH Challenge-Response Algorithms. `https://tools.ietf.org/id/draft-mraihi-mutual-oath-hotp-variants-11.html`, 2010.

[28] IETF. TOTP: Time-Based One-Time Password Algorithm. `https://tools.ietf.org/html/rfc6238`, 2011.

[29] IETF. The OAuth 2.0 Authorization Framework. `http://tools.ietf.org/html/rfc6749`, 2012.

[30] IETF. Proof Key for Code Exchange by OAuth Public Clients. `https://tools.ietf.org/html/rfc7636`, 2015.

[31] Internet-Draft. OAuth 2.0 Security Best Current Practice. `https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13`, 2019.

[32] iOS. Universal Links for Developers. https://developer.apple.com/ios/universal-links/, 2017.

[33] C. Jacomme and S. Kremer. An extensive formal analysis of multi-factor authentication protocols. In *31st IEEE Computer Security Foundations Symposium, CSF*, pages 1–15, 2018.

[34] L. Lamport. Password Authentication with Insecure Communication Communications. *Commun. ACM*, 24(11):770–772, 1981.

[35] G. Lowe. A Hierarchy of Authentication Specifications. In *10th IEEE Workshop on Computer Security Foundations*, 1997.

[36] S. Meier, B. Schmidt, C. Cremers, and D. A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 696–701, 2013.

[37] Ministero dell'Interno. Carta di Identità Elettronica. https://www.cartaidentita.interno.gov.it/, 2019.

[38] S. Mödersheim and L. Viganò. Secure Pseudonymous Channels. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS)*, pages 337–354, 2009.

[39] OASIS. SAML V2.0 technical overview. https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf, 2005.

[40] OAuth Working Group. OAuth 2.0 for Native Apps. https://tools.ietf.org/html/rfc8252, 2016.

[41] OIDF. OpenID Connect Core 1.0. http://openid.net/specs/openid-connect-core-1_0.html, 2014.

[42] V. Osmani, S. Forti, O. Mayora, and D. Conforti. Challenges and opportunities in evolving TreC personal health record platform. In *11th EAI International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pages 288–291, 2017.

[43] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh. Formal Verification of OAuth 2.0 Using Alloy Framework. In *Proceedings of the IEEE international conference on Communication Systems and Network Technologies (CSNT)*, pages 655–659, 2011.

[44] O. Pereira, F. Rochet, and C. Wiedling. Formal analysis of the FIDO 1.x protocol. In *Foundations and Practice of Security - 10th International Symposium, FPS*, pages 68–82, 2017.

[45] M. Pohl. 325,000 mobile health apps available in 2017 – android now the leading mhealth platform. https://research2guidance.com/325000-mobile-health-apps-available-in-2017/, 2017.

[46] G. Sciarretta, R. Carbone, S. Ranise, and A. Armando. Anatomy of the Facebook solution for mobile single sign-on: Security assessment and improvements. *Journal of Computers & Security*, 2017.

[47] G. Sciarretta, R. Carbone, S. Ranise, and L. Viganò. Design, Formal Specification and Analysis of Multi-Factor Authentication Solutions with a Single Sign-On Experience. In L. Bauer and R. Küsters, editors, *Principles of Security and Trust*, pages 188–213. Springer International Publishing, 2018.

[48] M. Shehab and F. Mohsen. Towards Enhancing the Security of OAuth Implementations in Smart Phones. In *IEEE International Conference on Mobile Services (MS)*, pages 39–46, 2014.

[49] F. Sinigaglia, R. Carbone, G. Costa, and N. Zannone. A survey on multi-factor authentication for online banking in the wild. *Computers & Security*, 2020.

[50] A. Sudhodanan, A. Armando, R. Carbone, and L. Compagna. Attack Patterns for Black-Box Security Testing of Multi-Party Web Applications. In *Proceedings of the 23nd Annual Network and Distributed System Security Symposium*, 2016.

[51] S. Sun and K. Beznosov. The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'12)*, 2012.

[52] L. Viganò. The spacios project: Secure provision and consumption in the internet of services. In *Sixth IEEE International Conference on Software Testing, Verification and Validation*, pages 497–498, 2013.

[53] D. von Oheimb and S. Mödersheim. ASLan++ — A Formal Security Specification Language for Distributed Systems. In *Proceedings of the 9th International Symposium on Formal Methods for Components and Objects (FMCO), revised papers*, LNCS 6957, pages 1–22. Springer, 2010.

[54] R. Wang, S. Chen, and X. Wang. Signing Me onto Your Accounts through Facebook and Google: A Traffic-Guided Security Study of Commercially Deployed Single-Sign-On Web Services. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 365–379, 2012.

[55] H. Yan, H. Fang, C. Kuka, and H. Zhu. Verification for OAuth Using ASLan++. In *Proceedings of 16th IEEE International Symposium on High Assurance Systems Engineering HASE*, pages 76–84, 2015.

[56] R. Yang, W. C. Lau, and T. Liu. Signing into One Billion Mobile App Accounts Effortlessly with OAuth2.0. In *Black Hat Europe*, 2016.

[57] Q. Ye, G. Bai, K. Wang, and J. S. Dong. Formal Analysis of a Single Sign-On Protocol Implementation for Android. In *Proceedings of the 20th ICECCS*, pages 90–99, 2015.

[58] Yubico. YubiKey NEO. `https://www.yubico.com/products/yubikey-hardware/yubikey-neo`, 2019.

A Abbreviations and Notations

Roles of *IdM*

UA	User Agent
SP	Service Provider
TP	Token Provider
IdP	Identity Provider
$HWToken$	Hardware Token
$IdTP$	An entity playing both the role of IdP and TP

Assumptions

sa	Security assumption
TA	Trust Assumption
CA	Channel Assumption
ActivA	Activation Assumption

Threat Model

$DT_{\square}$	Smartphone Device Thief
$DT_{\blacksquare}$	*eIDCard* Device Thief
SE	Social Engineer
MA	Malicious Application
AD	App Duplicator
SS	Shoulder Surfer
ES	Eavesdropping Software
OE	OTP Eavesdropping
TM	Set of all the attackers of our threat model $\|TM\| = 7$
$A = \{a_1, \ldots, a_j\} \subseteq TM$ with $\bar{A} \triangleq \bar{TM} \setminus A$	Set of attackers taken into consideration during the analysis
	Set of attackers for which we add protection during the analysis

Instance Factor

$IFactor_k$	Knowledge instance factor
$IFactor_o$	Ownership instance factor
IF_P	Set of instance factors involved in the communication protocol P (RM_{TOTP} or RM_{CR})
IF_P^{OTP}	Set of instance factors involved in P linked with OTP
$IF_P(a)$	Set of instance factors explicitly compromised by a in P
$IF_P(A) = \bigcup_{a_k \in A} IF_P(a_k)$	Set of instance factors explicitly compromised by a set of attackers A in P

Formalization

I	Initial knowledge
M	State transition system modeling RM_{TOTP} or RM_{CR}
$[\![sa]\!]$	Formal specification of sa
$[\![SA_M]\!] = \{[\![sa_1]\!], \ldots, [\![sa_m]\!]\}$	Set of security assumptions that formalizes $= \{[\![sa_1]\!], \ldots, [\![sa_m]\!]\}$ of M
$[\![a]\!]$	Formal specification of a
$[\![\bar{a}]\!]$	Formal specification of the protection to a
$Asms_A = [\![DY\|^A]\!] \cup [\![DY\|_{\bar{A}}]\!]$	Set of assumptions that formalizes the capabilities of a DY intruder: we empower a DY intruder with the properties formalized in $[\![DY\|^A]\!] = \{[\![a_1]\!], \ldots, [\![a_j]\!]\}$, while we limit his capabilities by adding the properties $[\![DY\|_{\bar{A}}]\!] = \{[\![\bar{a}_{j+1}]\!], \ldots, [\![\bar{a}_7]\!]\}$.